RTI Connext Traditional C++ API

Version 7.7.0

1 RTI Connext **1**

1.1 Available Documentation. . . . 1

1.1.1 The documents for the Core Libraries and Utilities are: . . . 2

1.1.2 The API Reference HTML documentation contains: . . . 2

1.2 Feedback and Support for this Release. . . . 2

2 Module Index **3**

2.1 Modules . . . 3

3 Namespace Index **9**

3.1 Namespace List . . . 9

4 Hierarchical Index **11**

4.1 Class Hierarchy . . . 11

5 Class Index **19**

5.1 Class List . . . 19

6 File Index **37**

6.1 File List . . . 37

7 Module Documentation **39**

7.1 RTI Connext Exceptions . . . 39

7.1.1 Detailed Description . . . 40

7.2 Clock Selection . . . 40

7.2.1 Available Clocks . . . 40

7.2.2 Clock Selection Strategy . . . 41

7.2.3 Configuring Clock Selection . . . 41

7.3 Domain Module . . . 41

7.3.1 Detailed Description . . . 42

7.4 DomainParticipantFactory . . . 42

7.4.1 Detailed Description . . . 43

7.4.2 Macro Definition Documentation . . . 43

7.4.2.1 DDSTheParticipantFactory . . . 44

7.4.3 Typedef Documentation . . . 44

7.4.3.1 DDSDomainParticipantFactory_RegisterTypeFunction . . . 44

7.4.4 Function Documentation . . . 44

7.4.4.1 DDS_DomainParticipantFactoryQos_equals() . . . 44

7.4.4.2 print() . . . 45

7.4.4.3 to_string() [1/6] . . . 45

7.4.4.4 to_string() [2/6] . . . 46

7.4.4.5 to_string() [3/6] . . . 47
7.4.4.6 to_string() [4/6] . . . 48
7.4.4.7 to_string() [5/6] . . . 48
7.4.4.8 to_string() [6/6] . . . 49
7.4.5 Variable Documentation . . . 50
7.4.5.1 DDS_PARTICIPANT_QOS_DEFAULT . . . 50
7.4.5.2 DDS_PARTICIPANT_CONFIG_PARAMS_DEFAULT . . . 51
7.5 DomainParticipants . . . 51
7.5.1 Detailed Description . . . 52
7.5.2 Typedef Documentation . . . 53
7.5.2.1 DDS_DomainId_t . . . 53
7.5.3 Function Documentation . . . 53
7.5.3.1 DDS_DomainParticipantQos_equals() . . . 53
7.5.3.2 print() . . . 53
7.5.3.3 to_string() [1/6] . . . 54
7.5.3.4 to_string() [2/6] . . . 55
7.5.3.5 to_string() [3/6] . . . 55
7.5.3.6 to_string() [4/6] . . . 56
7.5.3.7 to_string() [5/6] . . . 57
7.5.3.8 to_string() [6/6] . . . 58
7.5.4 Variable Documentation . . . 58
7.5.4.1 DDS_TOPIC_QOS_DEFAULT . . . 58
7.5.4.2 DDS_PUBLISHER_QOS_DEFAULT . . . 59
7.5.4.3 DDS_SUBSCRIBER_QOS_DEFAULT . . . 60
7.5.4.4 DDS_FLOW_CONTROLLER_PROPERTY_DEFAULT . . . 60
7.5.4.5 DDS_SQLFILTER_NAME . . . 61
7.5.4.6 DDS_STRINGMATCHFILTER_NAME . . . 61
7.6 Built-in Topics . . . 61
7.6.1 Detailed Description . . . 62
7.7 Topic Module . . . 63
7.7.1 Detailed Description . . . 64
7.8 Topics . . . 64
7.8.1 Detailed Description . . . 66
7.8.2 Typedef Documentation . . . 66
7.8.2.1 DDS_PrintFormatProperty . . . 66
7.8.3 Enumeration Type Documentation . . . 66
7.8.3.1 DDS_PrintFormatKind . . . 66
7.8.4 Function Documentation . . . 67
7.8.4.1 DDS_TopicQos_equals() . . . 67

7.8.4.2 print() . . . 67
7.8.4.3 to_string() [1/6] . . . 68
7.8.4.4 to_string() [2/6] . . . 68
7.8.4.5 to_string() [3/6] . . . 69
7.8.4.6 to_string() [4/6] . . . 70
7.8.4.7 to_string() [5/6] . . . 71
7.8.4.8 to_string() [6/6] . . . 71
7.8.5 Variable Documentation . . . 72
7.8.5.1 DDS_PRINT_FORMAT_PROPERTY_DEFAULT . . . 72
7.8.5.2 DDS_FILTERSAMPLEINFO_DEFAULT . . . 72
7.9 Zero Copy Transfer Over Shared Memory . . . 73
7.10 User Data Type Support . . . 73
7.10.1 Detailed Description . . . 74
7.10.2 Macro Definition Documentation . . . 74
7.10.2.1 DDS_TYPESUPPORT_CPP . . . 75
7.10.2.2 DDS_DATAWRITER_CPP . . . 75
7.10.2.3 DDS_DATAREADER_W_DATA_CONSISTENCY_CHECK . . . 76
7.10.3 Typedef Documentation . . . 76
7.10.3.1 DDS_InstanceHandle_t . . . 76
7.10.4 Function Documentation . . . 76
7.10.4.1 DDS_InstanceHandle_equals() . . . 76
7.10.4.2 DDS_InstanceHandle_compare() . . . 77
7.10.4.3 DDS_InstanceHandle_copy() . . . 77
7.10.4.4 DDS_InstanceHandle_is_nil() . . . 78
7.10.5 Variable Documentation . . . 78
7.10.5.1 DDS_HANDLE_NIL . . . 78
7.11 Type Code Support . . . 79
7.11.1 Detailed Description . . . 82
7.11.2 Accessing a Local ::DDS_TypeCode . . . 82
7.11.3 Accessing a Remote ::DDS_TypeCode . . . 83
7.11.4 Macro Definition Documentation . . . 83
7.11.4.1 DDS_TYPECODE_MEMBER_ID_INVALID . . . 83
7.11.4.2 DDS_TYPECODE_INDEX_INVALID . . . 83
7.11.4.3 DDS_TYPECODE_NOT_BITFIELD . . . 84
7.11.4.4 DDS_VM_NONE . . . 84
7.11.4.5 DDS_VM_CUSTOM . . . 84
7.11.4.6 DDS_VM_ABSTRACT . . . 84
7.11.4.7 DDS_VM_TRUNCATABLE . . . 85
7.11.4.8 DDS_PRIVATE_MEMBER . . . 85

7.11.4.9 DDS_PUBLIC_MEMBER . . . 85
7.11.4.10 DDS_TYPECODE_NONKEY_MEMBER . . . 86
7.11.4.11 DDS_TYPECODE_KEY_MEMBER . . . 86
7.11.4.12 DDS_TYPECODE_NONKEY_REQUIRED_MEMBER . . . 87
7.11.4.13 DDS_TypeCode_PrintFormat_INITIALIZER . . . 87
7.11.5 Typedef Documentation . . . 87
7.11.5.1 DDS_ValueModifier . . . 88
7.11.5.2 DDS_Visibility . . . 88
7.11.6 Enumeration Type Documentation . . . 88
7.11.6.1 DDS_TypeCodePrintFormatKind . . . 88
7.11.6.2 DDS_TCKind . . . 89
7.11.6.3 DDS_ExtensibilityKind . . . 89
7.11.7 Variable Documentation . . . 90
7.11.7.1 DDS_g_tc_null . . . 90
7.11.7.2 DDS_g_tc_short . . . 91
7.11.7.3 DDS_g_tc_long . . . 91
7.11.7.4 DDS_g_tc_ushort . . . 91
7.11.7.5 DDS_g_tc_ulong . . . 92
7.11.7.6 DDS_g_tc_float . . . 92
7.11.7.7 DDS_g_tc_double . . . 92
7.11.7.8 DDS_g_tc_boolean . . . 93
7.11.7.9 DDS_g_tc_char . . . 93
7.11.7.10 DDS_g_tc_octet . . . 93
7.11.7.11 DDS_g_tc_int8 . . . 94
7.11.7.12 DDS_g_tc_uint8 . . . 94
7.11.7.13 DDS_g_tc_longlong . . . 94
7.11.7.14 DDS_g_tc_ulonglong . . . 95
7.11.7.15 DDS_g_tc_longdouble . . . 95
7.11.7.16 DDS_g_tc_wchar . . . 95
7.12 Built-in Types . . . 95
7.12.1 Detailed Description . . . 96
7.12.2 Managing Memory for Builtin Types . . . 96
7.12.3 Typecodes for Builtin Types . . . 97
7.13 Built-in Topic's Trust Types . . . 99
7.13.1 Detailed Description . . . 100
7.13.2 Typedef Documentation . . . 100
7.13.2.1 DDS_ParticipantTrustAttributesMask . . . 100
7.13.2.2 DDS_PluginParticipantTrustAttributesMask . . . 100
7.13.2.3 DDS_ParticipantTrustProtectionInfo . . . 100

7.13.2.4 DDS_EndpointTrustAttributesMask . . . 101
7.13.2.5 DDS_PluginEndpointTrustAttributesMask . . . 101
7.13.2.6 DDS_VendorEndpointTrustAttributesMask . . . 101
7.13.2.7 DDS_EndpointTrustProtectionInfo . . . 101
7.13.2.8 DDS_TrustAlgorithmBit . . . 101
7.13.2.9 DDS_TrustAlgorithmSet . . . 101
7.13.2.10 DDS_TrustAlgorithmRequirements . . . 101
7.13.2.11 DDS_ParticipantTrustSignatureAlgorithmInfo . . . 102
7.13.2.12 DDS_ParticipantTrustKeyEstablishmentAlgorithmInfo . . . 102
7.13.2.13 DDS_ParticipantTrustInterceptorAlgorithmInfo . . . 102
7.13.2.14 DDS_ParticipantTrustAlgorithmInfo . . . 102
7.13.2.15 DDS_EndpointTrustInterceptorAlgorithmInfo . . . 102
7.13.2.16 DDS_EndpointTrustAlgorithmInfo . . . 103
7.14 Dynamic Data . . . 103
7.14.1 Detailed Description . . . 104
7.14.2 Macro Definition Documentation . . . 105
7.14.2.1 DDS_DYNAMIC_DATA_MEMBER_ID_UNSPECIFIED . . . 105
7.14.3 Typedef Documentation . . . 105
7.14.3.1 DDS_DynamicDataMemberId . . . 105
7.14.4 Variable Documentation . . . 105
7.14.4.1 DDS_DYNAMIC_DATA_PROPERTY_DEFAULT . . . 106
7.14.4.2 DDS_DYNAMIC_DATA_TYPE_PROPERTY_DEFAULT . . . 106
7.14.4.3 DDS_DYNAMIC_DATA_JSON_PARSER_PROPERTIES_DEFAULT . . . 106
7.15 Publication Module . . . 107
7.15.1 Detailed Description . . . 107
7.16 Publishers . . . 107
7.16.1 Detailed Description . . . 108
7.16.2 Function Documentation . . . 108
7.16.2.1 DDS_PublisherQos_equals() . . . 108
7.16.2.2 print() . . . 109
7.16.2.3 to_string() [1/6] . . . 109
7.16.2.4 to_string() [2/6] . . . 110
7.16.2.5 to_string() [3/6] . . . 111
7.16.2.6 to_string() [4/6] . . . 112
7.16.2.7 to_string() [5/6] . . . 112
7.16.2.8 to_string() [6/6] . . . 113
7.16.3 Variable Documentation . . . 114
7.16.3.1 DDS_DATAWRITER_QOS_DEFAULT . . . 114
7.16.3.2 DDS_DATAWRITER_QOS_USE_TOPIC_QOS . . . 115

7.17 Data Writers . . . 115
7.17.1 Detailed Description . . . 116
7.17.2 Function Documentation . . . 117
7.17.2.1 DDS_DataWriterQos_equals() . . . 117
7.17.2.2 print() . . . 117
7.17.2.3 to_string() [1/6] . . . 118
7.17.2.4 to_string() [2/6] . . . 118
7.17.2.5 to_string() [3/6] . . . 119
7.17.2.6 to_string() [4/6] . . . 120
7.17.2.7 to_string() [5/6] . . . 121
7.17.2.8 to_string() [6/6] . . . 121
7.18 Flow Controllers . . . 122
7.18.1 Detailed Description . . . 123
7.18.2 Enumeration Type Documentation . . . 123
7.18.2.1 DDS_FlowControllerSchedulingPolicy . . . 123
7.18.3 Variable Documentation . . . 125
7.18.3.1 DDS_DEFAULT_FLOW_CONTROLLER_NAME . . . 125
7.18.3.2 DDS_FIXED_RATE_FLOW_CONTROLLER_NAME . . . 126
7.18.3.3 DDS_ON_DEMAND_FLOW_CONTROLLER_NAME . . . 127
7.19 Subscription Module . . . 128
7.19.1 Detailed Description . . . 129
7.19.2 Access to data samples . . . 129
7.19.2.1 Data access patterns . . . 129
7.20 Subscribers . . . 130
7.20.1 Detailed Description . . . 131
7.20.2 Function Documentation . . . 131
7.20.2.1 DDS_SubscriberQos_equals() . . . 131
7.20.2.2 print() . . . 132
7.20.2.3 to_string() [1/6] . . . 132
7.20.2.4 to_string() [2/6] . . . 133
7.20.2.5 to_string() [3/6] . . . 134
7.20.2.6 to_string() [4/6] . . . 134
7.20.2.7 to_string() [5/6] . . . 135
7.20.2.8 to_string() [6/6] . . . 136
7.20.3 Variable Documentation . . . 136
7.20.3.1 DDS_DATAREADER_QOS_DEFAULT . . . 137
7.20.3.2 DDS_DATAREADER_QOS_USE_TOPIC_QOS . . . 137
7.21 DataReaders . . . 138
7.21.1 Detailed Description . . . 140

7.21.2 Enumeration Type Documentation . . . 140
7.21.2.1 DDS_SampleLostStatusKind . . . 140
7.21.2.2 DDS_SampleRejectedStatusKind . . . 144
7.21.3 Function Documentation . . . 146
7.21.3.1 DDS_DataReaderQos_equals() . . . 146
7.21.3.2 print() . . . 146
7.21.3.3 to_string() [1/6] . . . 147
7.21.3.4 to_string() [2/6] . . . 148
7.21.3.5 to_string() [3/6] . . . 148
7.21.3.6 to_string() [4/6] . . . 149
7.21.3.7 to_string() [5/6] . . . 150
7.21.3.8 to_string() [6/6] . . . 151
7.22 Read Conditions . . . 151
7.22.1 Detailed Description . . . 152
7.23 Query Conditions . . . 152
7.23.1 Detailed Description . . . 152
7.24 Data Samples . . . 152
7.24.1 Detailed Description . . . 153
7.24.2 Function Documentation . . . 153
7.24.2.1 DDS_CoherentSetInfo_equals() . . . 153
7.24.2.2 DDS_CoherentSetInfo_copy() . . . 154
7.24.2.3 DDS_SampleInfo_get_sample_identity() . . . 154
7.24.2.4 DDS_SampleInfo_get_related_sample_identity() . . . 154
7.25 Topic Queries . . . 155
7.25.1 Detailed Description . . . 156
7.25.2 Snapshot TopicQueries . . . 156
7.25.3 Reading TopicQuery Samples . . . 157
7.25.4 Debugging Topic Queries . . . 158
7.25.4.1 The Built-in ServiceRequest DataReader . . . 158
7.25.4.2 The on_service_request_accepted DataWriter Listener Callback . . . 158
7.25.5 Typedef Documentation . . . 158
7.25.5.1 DDS_TopicQuerySelection . . . 159
7.25.5.2 DDS_TopicQueryData . . . 159
7.25.6 Enumeration Type Documentation . . . 159
7.25.6.1 DDS_TopicQuerySelectionKind . . . 159
7.25.7 Variable Documentation . . . 160
7.25.7.1 DDS_TOPIC_QUERY_SELECTION_USE_READER_CONTENT_FILTER . . . 160
7.25.7.2 DDS_TOPIC_QUERY_SELECTION_SELECT_ALL . . . 160
7.26 Sample States . . . 160

7.26.1 Detailed Description . . . 161
7.26.2 Typedef Documentation . . . 161
7.26.2.1 DDS_SampleStateMask . . . 161
7.26.3 Enumeration Type Documentation . . . 161
7.26.3.1 DDS_SampleStateKind . . . 161
7.26.4 Variable Documentation . . . 162
7.26.4.1 DDS_ANY_SAMPLE_STATE . . . 162
7.27 View States . . . 162
7.27.1 Detailed Description . . . 163
7.27.2 Typedef Documentation . . . 163
7.27.2.1 DDS_ViewStateMask . . . 163
7.27.3 Enumeration Type Documentation . . . 163
7.27.3.1 DDS_ViewStateKind . . . 163
7.27.4 Variable Documentation . . . 164
7.27.4.1 DDS_ANY_VIEW_STATE . . . 164
7.28 Instance States . . . 164
7.28.1 Detailed Description . . . 165
7.28.2 Typedef Documentation . . . 165
7.28.2.1 DDS_InstanceStateMask . . . 165
7.28.3 Enumeration Type Documentation . . . 165
7.28.3.1 DDS_InstanceStateKind . . . 165
7.28.4 Variable Documentation . . . 166
7.28.4.1 DDS_ANY_INSTANCE_STATE . . . 166
7.28.4.2 DDS_NOT_ALIVE_INSTANCE_STATE . . . 166
7.29 Stream Kinds . . . 166
7.29.1 Detailed Description . . . 167
7.29.2 Typedef Documentation . . . 167
7.29.2.1 DDS_StreamKindMask . . . 167
7.29.3 Enumeration Type Documentation . . . 167
7.29.3.1 DDS_StreamKind . . . 167
7.30 Infrastructure Module . . . 167
7.30.1 Detailed Description . . . 169
7.30.2 Variable Documentation . . . 169
7.30.2.1 DDS_QOS_PRINT_ALL . . . 169
7.31 Built-in Sequences . . . 169
7.31.1 Detailed Description . . . 170
7.32 Multi-channel DataWriters . . . 170
7.32.1 What is a Multi-channel DataWriter? . . . 171
7.32.2 Configuration on the Writer Side . . . 171

7.32.3 Configuration on the Reader Side . . . 171
7.32.4 Reliability with Multi-Channel DataWriters . . . 171
7.32.4.1 Reliable Delivery . . . 171
7.32.4.2 Reliable Protocol Considerations . . . 172
7.33 Transports . . . 172
7.33.1 Detailed Description . . . 173
7.33.2 Overview . . . 173
7.33.3 Transport Aliases . . . 173
7.33.4 Transport Lifecycle . . . 174
7.33.5 Transport Class Attributes . . . 174
7.33.6 Transport Instance Attributes . . . 175
7.33.7 Transport Network Address . . . 176
7.33.8 Transport Send Route . . . 176
7.33.9 Transport Receive Route . . . 177
7.34 Installing Transport Plugins . . . 177
7.34.1 Detailed Description . . . 178
7.34.2 Loading Transport Plugins through Property QoS Policy of Domain Participant . . . 178
7.34.3 Typedef Documentation . . . 181
7.34.3.1 NDDS_Transport_Handle_t . . . 181
7.34.3.2 NDDS_Transport_create_plugin . . . 181
7.34.4 Function Documentation . . . 182
7.34.4.1 NDDS_Transport_Handle_is_nil() . . . 182
7.34.5 Variable Documentation . . . 182
7.34.5.1 NDDS_TRANSPORT_HANDLE_NIL . . . 182
7.35 Built-in Transport Plugins . . . 182
7.35.1 Detailed Description . . . 183
7.36 Creating New Transport Plugins . . . 183
7.37 Queries and Filters Syntax . . . 184
7.37.1 Syntax for DDS Queries and Filters . . . 184
7.37.2 SQL grammar in BNF . . . 184
7.37.3 Token expression . . . 185
7.37.4 String Parameters . . . 187
7.37.5 Type compatability in Predicate . . . 187
7.37.6 SQL Extension: Regular Expression Matching . . . 187
7.37.7 Character Encoding . . . 188
7.37.8 Unicode Normalization . . . 188
7.37.9 Examples . . . 189
7.38 Logging and Version . . . 189
7.38.1 Detailed Description . . . 190

7.39 General Utilities and Compliance Configuration . . . 190
7.39.1 Detailed Description . . . 190
7.40 Observability Framework . . . 190
7.40.1 Detailed Description . . . 190
7.41 Request-Reply Pattern . . . 191
7.41.1 Detailed Description . . . 191
7.42 Requester . . . 191
7.42.1 Detailed Description . . . 192
7.43 Replier . . . 192
7.43.1 Detailed Description . . . 192
7.44 Infrastructure . . . 192
7.44.1 Detailed Description . . . 193
7.44.2 Function Documentation . . . 194
7.44.2.1 make_valid_sample_iterator() . . . 194
7.44.2.2 move() . . . 195
7.45 Utilities . . . 195
7.45.1 Detailed Description . . . 196
7.46 Durability and Persistence . . . 196
7.46.1 Durable Writer History . . . 196
7.46.2 Durable Reader State . . . 197
7.46.3 Data Durability . . . 197
7.46.4 Durability and Persistence Based on Virtual GUID . . . 197
7.46.5 Configuring Durable Writer History . . . 198
7.46.6 Configuring Durable Reader State . . . 200
7.46.7 Configuring Data Durability . . . 201
7.47 Basic QoS . . . 201
7.48 DDS Specification QoS . . . 209
7.49 DDS Extension QoS . . . 213
7.50 Functional Categories . . . 252
7.50.1 Detailed Description . . . 253
7.51 Application Acknowledgement . . . 253
7.52 Application-Defined Metadata . . . 254
7.53 Application Startup . . . 256
7.54 Batching . . . 257
7.55 Concurrency . . . 258
7.56 Content Filtering . . . 260
7.57 Data Availability . . . 261
7.58 Debuggability . . . 266
7.59 Discovery . . . 271

7.60 Discovery Isolation . . . 277
7.61 Durability . . . 278
7.62 Flow Control . . . 282
7.63 Fragmentation . . . 284
7.64 Instance Lifecycle . . . 286
7.65 Sending Large Data . . . 290
7.66 Liveliness . . . 292
7.67 Logging . . . 294
7.68 Middleware Memory Management . . . 297
7.69 QoS Profile Management . . . 312
7.70 Reliability QoS . . . 314
7.71 Sample Memory Management QoS . . . 321
7.72 Sample Ordering . . . 324
7.73 Security . . . 325
7.74 Throughput . . . 327
7.75 Transports . . . 327
7.76 Topic Queries . . . 330
7.77 Type System . . . 331
7.78 QoS Mutability . . . 333
7.78.1 Detailed Description . . . 333
7.79 Immutable QoS . . . 333
7.80 Mutable QoS . . . 369
7.81 Other QoS Attributes . . . 375
7.81.1 Detailed Description . . . 375
7.82 Applied Per-Instance . . . 375
7.83 Discovery Metadata . . . 377
7.84 Matching . . . 382
7.85 Deprecated QoS . . . 384
7.86 System Properties . . . 389
7.86.1 System Properties List . . . 390
7.86.2 System Resource Consideration . . . 390
7.87 Configuring QoS Profiles with XML . . . 390
7.87.1 Loading QoS Profiles from XML Resources . . . 390
7.87.2 URL . . . 392
7.87.2.1 URL groups . . . 392
7.87.2.2 NDDS_QOS_PROFILES environment variable . . . 392
7.87.2.3 Built-In QoS Profiles . . . 392
7.88 Publication Example . . . 392
7.88.1 A typical publication example . . . 393

7.89 Subscription Example . . . 393
7.89.1 A typical subscription example . . . 394
7.90 Participant Use Cases . . . 394
7.90.1 Turning off auto-enable of newly created participant(s) . . . 395
7.90.2 Getting the factory . . . 395
7.90.3 Setting up a participant . . . 395
7.90.4 Tearing down a participant . . . 396
7.91 Topic Use Cases . . . 396
7.91.1 Registering a user data type . . . 396
7.91.2 Setting up a topic . . . 396
7.91.3 Tearing down a topic . . . 397
7.92 FlowController Use Cases . . . 397
7.92.1 Creating a flow controller . . . 397
7.92.2 Flow controlling a data writer . . . 397
7.92.3 Using the built-in flow controllers . . . 398
7.92.4 Shaping the network traffic for a particular transport . . . 399
7.92.5 Coalescing multiple samples in a single network packet . . . 399
7.93 Publisher Use Cases . . . 399
7.93.1 Setting up a publisher . . . 399
7.93.2 Tearing down a publisher . . . 399
7.94 DataWriter Use Cases . . . 400
7.94.1 Setting up a data writer . . . 400
7.94.2 Managing instances . . . 400
7.94.3 Sending data . . . 401
7.94.4 Tearing down a data writer . . . 401
7.95 Subscriber Use Cases . . . 401
7.95.1 Setting up a subscriber . . . 401
7.95.2 Set up subscriber to access received data . . . 402
7.95.3 Access received data via a subscriber . . . 402
7.95.4 Access received data coherently and/or in order . . . 403
7.95.5 Tearing down a subscriber . . . 403
7.96 DataReader Use Cases . . . 403
7.96.1 Setting up a data reader . . . 403
7.96.2 Managing instances . . . 404
7.96.3 Set up reader to access received data . . . 404
7.96.4 Access received data via a reader . . . 404
7.96.5 Taking data . . . 404
7.96.6 Reading data . . . 405
7.96.7 Tearing down a data reader . . . 406

7.97 Entity Use Cases . . . 406
7.97.1 Enabling an entity . . . 406
7.97.2 Checking if a status changed on an entity. . . . 406
7.97.3 Changing the QoS for an entity . . . 406
7.97.4 Changing the listener and enabling/disabling statuses associated with it . . . 407
7.97.5 Enabling/Disabling statuses associated with a status condition . . . 407
7.98 Waitset Use Cases . . . 408
7.98.1 Setting up a wait-set . . . 408
7.98.2 Waiting for condition(s) to trigger . . . 409
7.98.3 Tearing down a wait-set . . . 409
7.99 Transport Use Cases . . . 409
7.99.1 Changing the automatically registered built-in transports . . . 409
7.99.2 Changing the properties of the automatically registered builtin transports . . . 410
7.99.3 Creating a transport . . . 410
7.99.4 Deleting a transport . . . 410
7.99.5 Registering a transport with a participant . . . 411
7.99.6 Adding receive routes for a transport . . . 412
7.99.7 Adding send routes for a transport . . . 412
7.100 Filter Use Cases . . . 412
7.100.1 Introduction . . . 412
7.100.1.1 Overview of ContentFilteredTopic . . . 413
7.100.1.2 Overview of QueryCondition . . . 413
7.100.2 Filtering with ContentFilteredTopic . . . 414
7.100.3 Filtering with Query Conditions . . . 414
7.100.4 Filtering Performance . . . 415
7.101 Creating Custom Content Filters . . . 415
7.101.1 Introduction . . . 416
7.101.2 The Custom Content Filter API . . . 416
7.101.2.1 The compile function . . . 416
7.101.2.2 The evaluate function . . . 416
7.101.2.3 The finalize function . . . 417
7.101.3 Example Using C format strings . . . 417
7.101.3.1 Writing the Compile Function . . . 417
7.101.3.2 Writing the Evaluate Function . . . 417
7.101.3.3 Writing the Finalize Function . . . 417
7.101.3.4 Registering the Filter . . . 418
7.101.3.5 Unregistering the Filter . . . 418
7.102 Large Data Use Cases . . . 418
7.102.1 Introduction
 . . . 418

7.102.2 Writing Large Data . . . 419
7.102.3 Receiving Large Data . . . 419
7.103 Request-Reply Examples . . . 419
7.103.1 Request-Reply Examples . . . 419
7.103.2 Requester Creation . . . 421
7.103.3 Creating a Requester with optional parameters . . . 421
7.103.4 Basic Requester example . . . 421
7.103.5 Taking loaned samples . . . 422
7.103.6 Taking samples by copy . . . 422
7.103.7 Correlating requests and replies . . . 423
7.103.8 Basic Requester example using SampleRef . . . 424
7.103.9 Creating a Replier . . . 425
7.103.10 Basic Replier example . . . 425
7.103.11 SimpleReplier example . . . 426
7.103.12 Error handling example . . . 426
7.103.13 Configuring Request-Reply QoS profiles . . . 427
7.104 Documentation Roadmap . . . 430
7.105 Conventions . . . 431
7.105.1 Unsupported Features . . . 431
7.105.2 API Naming Conventions . . . 431
7.105.2.1 Structure & Class Names . . . 431
7.105.3 API Documentation Terms . . . 431
7.105.4 Stereotypes . . . 432
7.105.4.1 Basic Qos . . . 432
7.105.4.2 Extensions . . . 432
7.105.4.3 Experimental . . . 432
7.105.4.4 Types . . . 433
7.105.4.5 Method Parameters . . . 433
7.106 Using DDS:: Namespace . . . 433
7.106.1 DDS Namespace Support . . . 434
7.106.2 DDS Namespace and Primitive Types . . . 434
7.107 RTI Connext DDS API Reference . . . 434
7.107.1 Detailed Description . . . 435
7.107.2 Overview . . . 436
7.107.3 Conceptual Model . . . 436
7.107.4 Modules . . . 438
7.108 Additional RTI Connext Communication Patterns . . . 438
7.108.1 Detailed Description . . . 438
7.109 Programming How-To's . . . 439

7.109.1 Detailed Description . . . 439
7.110 Interface . . . 440
7.110.1 Detailed Description . . . 440
7.110.2 Enumeration Type Documentation . . . 440
7.110.2.1 NDDS_Transport_Interface_Status_t . . . 440
7.111 Transport Plugins Configuration . . . 441
7.111.1 Detailed Description . . . 442
7.111.2 Macro Definition Documentation . . . 443
7.111.2.1 NDDS_TRANSPORT_PORT_INVALID . . . 443
7.111.2.2 NDDS_TRANSPORT_UUID_SIZE . . . 443
7.111.2.3 NDDS_TRANSPORT_LENGTH_UNLIMITED . . . 443
7.111.2.4 NDDS_TRANSPORT_INTERFACE_RANK_UNKNOWN . . . 443
7.111.2.5 NDDS_TRANSPORT_UUID_UNKNOWN . . . 443
7.111.2.6 NDDS_TRANSPORT_ALLOCATION_SETTINGS_MAX_COUNT_UNLIMITED . . . 444
7.111.2.7 NDDS_TRANSPORT_ALLOCATION_SETTINGS_INCREMENTAL_COUNT_AUTOMATIC 444
7.111.2.8 NDDS_TRANSPORT_ALLOCATION_SETTINGS_DEFAULT . . . 444
7.111.2.9 NDDS_TRANSPORT_CLASSID_INVALID . . . 444
7.111.2.10 NDDS_TRANSPORT_CLASSID_UDPv4 . . . 445
7.111.2.11 NDDS_TRANSPORT_CLASSID_SHMEM . . . 445
7.111.2.12 NDDS_TRANSPORT_CLASSID_SHMEM_510 . . . 445
7.111.2.13 NDDS_TRANSPORT_CLASSID_UDPv6 . . . 445
7.111.2.14 NDDS_TRANSPORT_CLASSID_UDPv6_510 . . . 445
7.111.2.15 NDDS_TRANSPORT_CLASSID_TCPV4_LAN . . . 445
7.111.2.16 NDDS_TRANSPORT_CLASSID_TCPV4_WAN . . . 446
7.111.2.17 NDDS_TRANSPORT_CLASSNAME_TCPV4_WAN . . . 446
7.111.2.18 NDDS_TRANSPORT_CLASSID_TLSV4_LAN . . . 446
7.111.2.19 NDDS_TRANSPORT_CLASSID_TLSV4_WAN . . . 446
7.111.2.20 NDDS_TRANSPORT_CLASSID_UDPv4_WAN . . . 446
7.111.2.21 NDDS_TRANSPORT_CLASSID_RESERVED_RANGE . . . 446
7.111.2.22 NDDS_TRANSPORT_PROPERTY_BIT_BUFFER_ALWAYS_LOANED . . . 447
7.111.2.23 NDDS_TRANSPORT_PROPERTY_GATHER_SEND_BUFFER_COUNT_MIN . . . 447
7.111.3 Typedef Documentation . . . 447
7.111.3.1 NDDS_Transport_Port_t . . . 447
7.111.3.2 NDDS_Transport_ClassId_t . . . 448
7.112 Transport Address . . . 448
7.112.1 Detailed Description . . . 449
7.112.2 Macro Definition Documentation . . . 449
7.112.2.1 NDDS_TRANSPORT_ADDRESS_INVALID_INITIALIZER . . . 450
7.112.2.2 NDDS_TRANSPORT_ADDRESS_STRING_BUFFER_SIZE . . . 450

7.112.3 Function Documentation . . . 450
7.112.3.1 NDDS_Transport_Address_to_string() . . . 450
7.112.3.2 NDDS_Transport_Address_to_string_with_protocol_family_format() . . . 451
7.112.3.3 NDDS_Transport_Address_from_string() . . . 452
7.112.3.4 NDDS_Transport_Address_print() . . . 452
7.112.3.5 NDDS_Transport_Address_is_ipv4() . . . 452
7.112.3.6 NDDS_Transport_Address_is_multicast() . . . 453
7.112.4 Variable Documentation . . . 453
7.112.4.1 NDDS_TRANSPORT_ADDRESS_INVALID . . . 453
7.113 UDP Transport Plugin definitions . . . 454
7.113.1 Detailed Description . . . 454
7.113.2 Macro Definition Documentation . . . 454
7.113.2.1 NDDS_TRANSPORT_UDP_PROPERTIES_BITMAP_DEFAULT . . . 455
7.113.2.2 NDDS_TRANSPORT_UDP_GATHER_SEND_BUFFER_COUNT_MAX_DEFAULT . . . 455
7.113.2.3 NDDS_TRANSPORT_UDP_SOCKET_BUFFER_SIZE_OS_MAX . . . 455
7.113.2.4 NDDS_TRANSPORT_UDP_SOCKET_BUFFER_SIZE_OS_DEFAULT . . . 455
7.113.2.5 NDDS_TRANSPORT_UDP_SEND_SOCKET_BUFFER_SIZE_DEFAULT . . . 456
7.113.2.6 NDDS_TRANSPORT_UDP_RECV_SOCKET_BUFFER_SIZE_DEFAULT . . . 456
7.113.2.7 NDDS_TRANSPORT_UDP_MULTICAST_TTL_DEFAULT . . . 456
7.113.3 Typedef Documentation . . . 456
7.113.3.1 NDDS_Transport_UDP_Port . . . 456
7.114 Shared Memory Transport . . . 456
7.114.1 Detailed Description . . . 457
7.114.2 Compatibility of Sender and Receiver Transports . . . 457
7.114.3 Crashing and Restarting Programs . . . 458
7.114.4 Shared Resource Keys . . . 458
7.114.5 Creating and Registering Shared Memory Transport Plugin . . . 458
7.114.6 Shared Memory Transport Property Names in Property QoS Policy of Domain Participant . . . 459
7.114.7 Macro Definition Documentation . . . 460
7.114.7.1 NDDS_TRANSPORT_SHMEM_ADDRESS_BIT_COUNT . . . 460
7.114.7.2 NDDS_TRANSPORT_SHMEM_PROPERTIES_BITMAP_DEFAULT . . . 460
7.114.7.3 NDDS_TRANSPORT_SHMEM_GATHER_SEND_BUFFER_COUNT_MAX_DEFAULT 460
7.114.7.4 NDDS_TRANSPORT_SHMEM_MESSAGE_SIZE_MAX_DEFAULT . . . 461
7.114.7.5 NDDS_TRANSPORT_SHMEM_RECEIVED_MESSAGE_COUNT_MAX_DEFAULT . . . 461
7.114.7.6 NDDS_TRANSPORT_SHMEM_RECEIVE_BUFFER_SIZE_DEFAULT . . . 461
7.114.7.7 NDDS_TRANSPORT_SHMEM_MAJOR_AFTER_BUG_14240_FIX . . . 461
7.114.7.8 NDDS_TRANSPORT_SHMEM_PROPERTY_DEFAULT . . . 461
7.114.8 Function Documentation . . . 461
7.114.8.1 NDDS_Transport_Shmem_new() . . . 461

7.114.8.2 NDDS_Transport_Shmem_create() . . . 462
7.115 UDPv4 Transport . . . 462
7.115.1 Detailed Description . . . 464
7.115.2 UDPv4 Transport Property Names in Property QoS Policy of Domain Participant . . . 465
7.115.3 Macro Definition Documentation . . . 467
7.115.3.1 NDDS_TRANSPORT_UDPV4_ADDRESS_BIT_COUNT . . . 467
7.115.3.2 NDDS_TRANSPORT_UDPV4_WAN_ADDRESS_BIT_COUNT . . . 467
7.115.3.3 NDDS_TRANSPORT_UDPV4_PROPERTIES_BITMAP_DEFAULT . . . 467
7.115.3.4 NDDS_TRANSPORT_UDPV4_GATHER_SEND_BUFFER_COUNT_MAX_DEFAULT . 467
7.115.3.5 NDDS_TRANSPORT_UDPV4_SOCKET_BUFFER_SIZE_OS_MAX . . . 468
7.115.3.6 NDDS_TRANSPORT_UDPV4_SOCKET_BUFFER_SIZE_OS_DEFAULT . . . 468
7.115.3.7 NDDS_TRANSPORT_UDPV4_SEND_SOCKET_BUFFER_SIZE_DEFAULT . . . 468
7.115.3.8 NDDS_TRANSPORT_UDPV4_RECV_SOCKET_BUFFER_SIZE_DEFAULT . . . 468
7.115.3.9 NDDS_TRANSPORT_UDPV4_PAYLOAD_SIZE_MAX . . . 468
7.115.3.10 NDDS_TRANSPORT_UDPV4_MESSAGE_SIZE_MAX_DEFAULT . . . 469
7.115.3.11 NDDS_TRANSPORT_UDPV4_MULTICAST_TTL_DEFAULT . . . 469
7.115.3.12 NDDS_TRANSPORT_UDPV4_BLOCKING_NEVER . . . 469
7.115.3.13 NDDS_TRANSPORT_UDPV4_BLOCKING_ALWAYS . . . 469
7.115.3.14 NDDS_TRANSPORT_UDPV4_BLOCKING_DEFAULT . . . 469
7.115.3.15 NDDS_TRANSPORT_UDPV4_PROPERTY_DEFAULT . . . 469
7.115.3.16 NDDS_Transport_UDPv4_string_to_address_cEA . . . 469
7.115.4 Function Documentation . . . 470
7.115.4.1 NDDS_Transport_UDPv4_new() . . . 470
7.115.4.2 NDDS_Transport_UDPv4_create() . . . 471
7.115.4.3 NDDS_Transport_UDPv4_create_from_properties_with_prefix() . . . 472
7.116 Real-Time WAN Transport . . . 472
7.116.1 Detailed Description . . . 473
7.116.2 Real-Time WAN Transport Property . . . 474
7.116.3 Macro Definition Documentation . . . 476
7.116.3.1 NDDS_TRANSPORT_UDPV4_WAN_MESSAGE_SIZE_MAX_DEFAULT . . . 476
7.116.3.2 NDDS_TRANSPORT_UDPV4_WAN_PROPERTY_DEFAULT . . . 476
7.116.4 Function Documentation . . . 477
7.116.4.1 NDDS_Transport_UDPv4_WAN_new() . . . 477
7.116.4.2 NDDS_Transport_UDPv4_WAN_create() . . . 477
7.116.4.3 NDDS_Transport_UDPv4_WAN_create_from_properties_with_prefix() . . . 478
7.117 UDPv6 Transport . . . 479
7.117.1 Detailed Description . . . 480
7.117.2 UDPv6 Transport Property Names in Property QoS Policy of Domain Participant . . . 481
7.117.3 Macro Definition Documentation . . . 483

7.117.3.1 NDDS_TRANSPORT_UDPV6_ADDRESS_BIT_COUNT . . . 483
7.117.3.2 NDDS_TRANSPORT_UDPV6_PROPERTIES_BITMAP_DEFAULT . . . 483
7.117.3.3 NDDS_TRANSPORT_UDPV6_GATHER_SEND_BUFFER_COUNT_MAX_DEFAULT . 483
7.117.3.4 NDDS_TRANSPORT_UDPV6_SOCKET_BUFFER_SIZE_OS_MAX . . . 484
7.117.3.5 NDDS_TRANSPORT_UDPV6_SOCKET_BUFFER_SIZE_OS_DEFAULT . . . 484
7.117.3.6 NDDS_TRANSPORT_UDPV6_SEND_SOCKET_BUFFER_SIZE_DEFAULT . . . 484
7.117.3.7 NDDS_TRANSPORT_UDPV6_RECV_SOCKET_BUFFER_SIZE_DEFAULT . . . 484
7.117.3.8 NDDS_TRANSPORT_UDPV6_PAYLOAD_SIZE_MAX . . . 484
7.117.3.9 NDDS_TRANSPORT_UDPV6_MESSAGE_SIZE_MAX_DEFAULT . . . 485
7.117.3.10 NDDS_TRANSPORT_UDPV6_MULTICAST_TTL_DEFAULT . . . 485
7.117.3.11 NDDS_TRANSPORT_UDPV6_BLOCKING_NEVER . . . 485
7.117.3.12 NDDS_TRANSPORT_UDPV6_BLOCKING_ALWAYS . . . 485
7.117.3.13 NDDS_Transport_UDPv6_string_to_address_cEA . . . 485
7.117.4 Function Documentation . . . 486
7.117.4.1 NDDS_Transport_UDPv6_new() . . . 486
7.117.4.2 NDDS_Transport_UDPv6_create() . . . 487
7.117.4.3 NDDS_Transport_UDPv6_create_from_properties_with_prefix() . . . 488
7.118 AsyncWaitSet . . . 488
7.118.1 Detailed Description . . . 489
7.118.2 Variable Documentation . . . 489
7.118.2.1 DDS_ASYNC_WAITSET_PROPERTY_DEFAULT . . . 489
7.118.2.2 COMPLETION_TOKEN_USE_IMPLICIT_AND_WAIT . . . 489
7.118.2.3 COMPLETION_TOKEN_IGNORE . . . 490
7.119 Participant Built-in Topics . . . 490
7.119.1 Detailed Description . . . 491
7.119.2 Typedef Documentation . . . 491
7.119.2.1 DDS_ParticipantBuiltinTopicData . . . 491
7.119.3 Variable Documentation . . . 491
7.119.3.1 DDS_PARTICIPANT_TOPIC_NAME . . . 491
7.120 Topic Built-in Topics . . . 492
7.120.1 Detailed Description . . . 492
7.120.2 Typedef Documentation . . . 492
7.120.2.1 DDS_TopicBuiltinTopicData . . . 492
7.120.3 Variable Documentation . . . 493
7.120.3.1 DDS_TOPIC_TOPIC_NAME . . . 493
7.121 Publication Built-in Topics . . . 493
7.121.1 Detailed Description . . . 494
7.121.2 Typedef Documentation . . . 494
7.121.2.1 DDS_PublicationBuiltinTopicData . . . 494

7.121.3 Variable Documentation . . . 494
7.121.3.1 DDS_PUBLICATION_TOPIC_NAME . . . 494
7.122 Subscription Built-in Topics . . . 494
7.122.1 Detailed Description . . . 495
7.122.2 Typedef Documentation . . . 495
7.122.2.1 DDS_SubscriptionBuiltinTopicData . . . 495
7.122.3 Variable Documentation . . . 496
7.122.3.1 DDS_SUBSCRIPTION_TOPIC_NAME . . . 496
7.123 ServiceRequest Built-in Topic . . . 496
7.123.1 Detailed Description . . . 497
7.123.2 Typedef Documentation . . . 497
7.123.2.1 DDS_ServiceRequest . . . 497
7.123.3 Variable Documentation . . . 498
7.123.3.1 DDS_UNKNOWN_SERVICE_REQUEST_ID . . . 498
7.123.3.2 DDS_TOPIC_QUERY_SERVICE_REQUEST_ID . . . 498
7.123.3.3 DDS_LOCATOR_REACHABILITY_SERVICE_REQUEST_ID . . . 498
7.123.3.4 DDS_INSTANCE_STATE_SERVICE_REQUEST_ID . . . 498
7.123.3.5 DDS_MONITORING_LIBRARY_COMMAND_SERVICE_REQUEST_ID . . . 498
7.123.3.6 DDS_MONITORING_LIBRARY_REPLY_SERVICE_REQUEST_ID . . . 499
7.123.3.7 DDS_SERVICE_REQUEST_TOPIC_NAME . . . 499
7.124 Common types and functions . . . 499
7.124.1 Detailed Description . . . 501
7.124.2 Macro Definition Documentation . . . 501
7.124.2.1 DDS_LOCATOR_ADDRESS_LENGTH_MAX . . . 501
7.124.2.2 DDS_PROTOCOLVERSION_1_0 . . . 502
7.124.2.3 DDS_PROTOCOLVERSION_1_1 . . . 502
7.124.2.4 DDS_PROTOCOLVERSION_1_2 . . . 502
7.124.2.5 DDS_PROTOCOLVERSION_2_0 . . . 502
7.124.2.6 DDS_PROTOCOLVERSION_2_1 . . . 502
7.124.2.7 DDS_PROTOCOLVERSION . . . 502
7.124.2.8 DDS_VENDOR_ID_LENGTH_MAX . . . 503
7.124.2.9 DDS_PRODUCTVERSION_UNKNOWN . . . 503
7.124.3 Typedef Documentation . . . 503
7.124.3.1 DDS_Locator_t . . . 503
7.124.3.2 DDS_ProtocolVersion_t . . . 503
7.124.3.3 DDS_BuiltinTopicKey_t . . . 504
7.124.4 Function Documentation . . . 504
7.124.4.1 DDS_BuiltinTopicKey_equals() . . . 504
7.124.4.2 DDS_BuiltinTopicKey_copy() . . . 504

7.124.4.3 DDS_BuiltinTopicKey_to_guid() . . . 505
7.124.4.4 DDS_BuiltinTopicKey_from_guid() . . . 505
7.124.4.5 DDS_BuiltinTopicKey_to_instance_handle() . . . 505
7.124.4.6 DDS_BuiltinTopicKey_from_instance_handle() . . . 506
7.124.5 Variable Documentation . . . 506
7.124.5.1 DDS_LOCATOR_INVALID . . . 506
7.124.5.2 DDS_LOCATOR_KIND_INVALID . . . 506
7.124.5.3 DDS_LOCATOR_PORT_INVALID . . . 507
7.124.5.4 DDS_LOCATOR_ADDRESS_INVALID . . . 507
7.124.5.5 DDS_LOCATOR_KIND_UDPv4 . . . 507
7.124.5.6 DDS_LOCATOR_KIND_UDPv4_WAN . . . 507
7.124.5.7 DDS_LOCATOR_KIND_SHMEM . . . 507
7.124.5.8 DDS_LOCATOR_KIND_SHMEM_510 . . . 507
7.124.5.9 DDS_LOCATOR_KIND_UDPv6 . . . 508
7.124.5.10 DDS_LOCATOR_KIND_UDPv6_510 . . . 508
7.124.5.11 DDS_LOCATOR_KIND_RESERVED . . . 508
7.124.5.12 value . . . 508
7.124.5.13 DDS_BUILTIN_TOPIC_KEY_UNKNOWN . . . 508
7.125 String Built-in Type . . . 508
7.125.1 Detailed Description . . . 509
7.126 KeyedString Built-in Type . . . 509
7.126.1 Detailed Description . . . 509
7.126.2 Typedef Documentation . . . 509
7.126.2.1 DDS_KeyedString . . . 510
7.127 Octets Built-in Type . . . 510
7.127.1 Detailed Description . . . 510
7.127.2 Typedef Documentation . . . 510
7.127.2.1 DDS_Octets . . . 510
7.128 KeyedOctets Built-in Type . . . 511
7.128.1 Detailed Description . . . 511
7.128.2 Typedef Documentation . . . 511
7.128.2.1 DDS_KeyedOctets . . . 511
7.129 DDS-Specific Primitive Types . . . 511
7.129.1 Detailed Description . . . 513
7.129.2 Macro Definition Documentation . . . 513
7.129.2.1 DDS_BOOLEAN_TRUE . . . 513
7.129.2.2 DDS_BOOLEAN_FALSE . . . 513
7.129.3 Typedef Documentation . . . 513
7.129.3.1 DDS_Char . . . 514

7.129.3.2 DDS_Wchar . . . 514
7.129.3.3 DDS_Octet . . . 514
7.129.3.4 DDS_UInt8 . . . 514
7.129.3.5 DDS_Int8 . . . 514
7.129.3.6 DDS_Short . . . 514
7.129.3.7 DDS_UnsignedShort . . . 515
7.129.3.8 DDS_Long . . . 515
7.129.3.9 DDS_UnsignedLong . . . 515
7.129.3.10 DDS_LongLong . . . 515
7.129.3.11 DDS_UnsignedLongLong . . . 515
7.129.3.12 DDS_Float . . . 515
7.129.3.13 DDS_Double . . . 516
7.129.3.14 DDS_LongDouble . . . 516
7.129.3.15 DDS_Boolean . . . 516
7.129.3.16 DDS_Enum . . . 516
7.130 Time Support . . . 517
7.130.1 Detailed Description . . . 518
7.130.2 Macro Definition Documentation . . . 518
7.130.2.1 DDS_TIME_ZERO . . . 518
7.130.3 Function Documentation . . . 519
7.130.3.1 from_micros() [1/2] . . . 519
7.130.3.2 from_millis() [1/2] . . . 519
7.130.3.3 from_nanos() [1/2] . . . 519
7.130.3.4 from_seconds() [1/2] . . . 519
7.130.3.5 from_micros() [2/2] . . . 519
7.130.3.6 from_millis() [2/2] . . . 520
7.130.3.7 from_nanos() [2/2] . . . 520
7.130.3.8 from_seconds() [2/2] . . . 520
7.130.3.9 DDS_Time_is_zero() . . . 520
7.130.3.10 DDS_Time_is_invalid() . . . 520
7.130.3.11 DDS_Duration_is_infinite() . . . 521
7.130.3.12 DDS_Duration_is_auto() . . . 521
7.130.3.13 DDS_Duration_is_zero() . . . 521
7.130.4 Variable Documentation . . . 521
7.130.4.1 DDS_TIME_MAX . . . 521
7.130.4.2 DDS_TIME_INVALID_SEC . . . 522
7.130.4.3 DDS_TIME_INVALID_NSEC . . . 522
7.130.4.4 DDS_TIME_INVALID . . . 522
7.130.4.5 DDS_DURATION_INFINITE_SEC . . . 522

7.130.4.6 DDS_DURATION_INFINITE_NSEC . . . 522
7.130.4.7 DDS_DURATION_INFINITE . . . 522
7.130.4.8 DDS_DURATION_AUTO_SEC . . . 523
7.130.4.9 DDS_DURATION_AUTO_NSEC . . . 523
7.130.4.10 DDS_DURATION_AUTO . . . 523
7.130.4.11 DDS_DURATION_ZERO_SEC . . . 523
7.130.4.12 DDS_DURATION_ZERO_NSEC . . . 523
7.130.4.13 DDS_DURATION_ZERO . . . 523
7.131 GUID Support . . . 524
7.131.1 Detailed Description . . . 525
7.131.2 Typedef Documentation . . . 525
7.131.2.1 DDS_RTPS_GuidPrefix_t . . . 525
7.131.2.2 DDS_RTPS_EntityId_t . . . 525
7.131.2.3 DDS_RTPS_GUID_t . . . 525
7.131.2.4 DDS_GUID_t . . . 525
7.131.3 Function Documentation . . . 525
7.131.3.1 DDS_GUID_equals() . . . 525
7.131.3.2 DDS_GUID_compare() . . . 526
7.131.3.3 DDS_GUID_copy() . . . 526
7.131.4 Variable Documentation . . . 527
7.131.4.1 DDS_GUID_AUTO . . . 527
7.131.4.2 DDS_GUID_UNKNOWN . . . 527
7.131.4.3 DDS_GUID_ZERO . . . 527
7.132 Sequence Number Support . . . 527
7.132.1 Detailed Description . . . 528
7.132.2 Typedef Documentation . . . 528
7.132.2.1 DDS_SequenceNumber_t . . . 528
7.132.3 Variable Documentation . . . 528
7.132.3.1 DDS_SEQUENCE_NUMBER_UNKNOWN . . . 528
7.132.3.2 DDS_SEQUENCE_NUMBER_ZERO . . . 529
7.132.3.3 DDS_SEQUENCE_NUMBER_MAX . . . 529
7.132.3.4 DDS_AUTO_SEQUENCE_NUMBER . . . 529
7.133 Exception Codes . . . 529
7.133.1 Detailed Description . . . 530
7.133.2 Enumeration Type Documentation . . . 530
7.133.2.1 DDS_ExceptionCode_t . . . 530
7.134 Return Codes . . . 531
7.134.1 Detailed Description . . . 531
7.134.2 Standard Return Codes . . . 532

7.134.3 Enumeration Type Documentation . . . 532
7.134.3.1 DDS_ReturnCode_t . . . 532
7.135 Status Kinds . . . 533
7.135.1 Detailed Description . . . 534
7.135.2 Changes in Status . . . 535
7.135.2.1 Changes in plain communication status . . . 536
7.135.2.2 Changes in read communication status . . . 536
7.135.3 Macro Definition Documentation . . . 537
7.135.3.1 DDS_STATUS_MASK_NONE . . . 537
7.135.3.2 DDS_STATUS_MASK_ALL . . . 537
7.135.4 Typedef Documentation . . . 537
7.135.4.1 DDS_StatusMask . . . 538
7.135.5 Enumeration Type Documentation . . . 538
7.135.5.1 DDS_StatusKind . . . 538
7.136 Thread Settings . . . 546
7.136.1 Detailed Description . . . 547
7.136.2 Macro Definition Documentation . . . 547
7.136.2.1 DDS_THREAD_SETTINGS_KIND_MASK_DEFAULT . . . 547
7.136.3 Typedef Documentation . . . 547
7.136.3.1 DDS_ThreadSettingsKindMask . . . 547
7.136.4 Enumeration Type Documentation . . . 548
7.136.4.1 DDS_ThreadSettingsKind . . . 548
7.136.4.2 DDS_ThreadSettingsCpuRotationKind . . . 548
7.136.5 Controlling CPU Core Affinity for RTI Threads . . . 548
7.137 QoS . . . 549
7.137.1 Detailed Description . . . 549
7.138 All QoS Policies . . . 549
7.138.1 Detailed Description . . . 554
7.138.2 Specifying QoS on entities . . . 555
7.138.3 QoS compatibility . . . 556
7.138.4 Macro Definition Documentation . . . 556
7.138.4.1 DDS_QosPrintFormat_INITIALIZER . . . 557
7.138.4.2 DDS_QOS_POLICY_COUNT . . . 557
7.138.5 Enumeration Type Documentation . . . 557
7.138.5.1 DDS_QosPolicyId_t . . . 557
7.139 QoS by Category . . . 560
7.139.1 Detailed Description . . . 560
7.140 ASYNCHRONOUS_PUBLISHER . . . 561
7.140.1 Detailed Description . . . 561

7.140.2 Variable Documentation . . . 561
7.140.2.1 DDS_ASYNCHRONOUSPUBLISHER_QOS_POLICY_NAME . . . 561
7.141 AVAILABILITY . . . 561
7.141.1 Detailed Description . . . 562
7.141.2 Variable Documentation . . . 562
7.141.2.1 DDS_AVAILABILITY_QOS_POLICY_NAME . . . 562
7.142 BATCH . . . 562
7.142.1 Detailed Description . . . 562
7.142.2 Variable Documentation . . . 563
7.142.2.1 DDS_BATCH_QOS_POLICY_NAME . . . 563
7.143 DATABASE . . . 563
7.143.1 Detailed Description . . . 563
7.143.2 Variable Documentation . . . 563
7.143.2.1 DDS_DATABASE_QOS_POLICY_NAME . . . 563
7.144 DATA_READER_PROTOCOL . . . 564
7.144.1 Detailed Description . . . 564
7.144.2 Variable Documentation . . . 564
7.144.2.1 DDS_DATAREADERPROTOCOL_QOS_POLICY_NAME . . . 564
7.145 DATA_READER_RESOURCE_LIMITS . . . 564
7.145.1 Detailed Description . . . 565
7.145.2 Enumeration Type Documentation . . . 565
7.145.2.1 DDS_DataReaderInstanceRemovalKind . . . 565
7.145.3 Variable Documentation . . . 567
7.145.3.1 DDS_DATAREADERRESOURCELIMITS_QOS_POLICY_NAME . . . 567
7.145.3.2 DDS_AUTO_MAX_TOTAL_INSTANCES . . . 567
7.146 DATA_REPRESENTATION . . . 567
7.146.1 Detailed Description . . . 568
7.146.2 Typedef Documentation . . . 568
7.146.2.1 DDS_DataRepresentationId_t . . . 568
7.146.3 Variable Documentation . . . 568
7.146.3.1 DDS_XCDR_DATA_REPRESENTATION . . . 568
7.146.3.2 DDS_XML_DATA_REPRESENTATION . . . 569
7.146.3.3 DDS_XCDR2_DATA_REPRESENTATION . . . 569
7.146.3.4 DDS_AUTO_DATA_REPRESENTATION . . . 569
7.146.3.5 DDS_DATA_REPRESENTATION_QOS_POLICY_NAME . . . 569
7.147 Compression Settings . . . 570
7.147.1 Detailed Description . . . 571
7.147.2 Macro Definition Documentation . . . 571
7.147.2.1 DDS_COMPRESSION_ID_MASK_NONE . . . 571

7.147.2.2 DDS_COMPRESSION_ID_MASK_ALL . . . 571
7.147.2.3 DDS_COMPRESSION_LEVEL_BEST_COMPRESSION . . . 571
7.147.2.4 DDS_COMPRESSION_LEVEL_BEST_SPEED . . . 572
7.147.2.5 DDS_COMPRESSION_LEVEL_DEFAULT . . . 572
7.147.2.6 DDS_COMPRESSION_THRESHOLD_DEFAULT . . . 572
7.147.2.7 DDS_COMPRESSION_ID_MASK_PUBLICATION_DEFAULT . . . 572
7.147.2.8 DDS_COMPRESSION_ID_MASK_SUBSCRIPTION_DEFAULT . . . 573
7.147.3 Typedef Documentation . . . 573
7.147.3.1 DDS_CompressionId_t . . . 573
7.147.3.2 DDS_CompressionIdMask . . . 573
7.147.4 Variable Documentation . . . 573
7.147.4.1 DDS_COMPRESSION_ID_ZLIB . . . 573
7.147.4.2 DDS_COMPRESSION_ID_BZIP2 . . . 574
7.147.4.3 DDS_COMPRESSION_ID_LZ4 . . . 574
7.148 DATA_TAG . . . 574
7.148.1 Detailed Description . . . 575
7.148.2 Typedef Documentation . . . 575
7.148.2.1 DDS_DataTagQosPolicy . . . 575
7.148.3 Usage . . . 575
7.148.4 Function Documentation . . . 576
7.148.4.1 get_number_of_tags() . . . 576
7.148.4.2 assert_tag() . . . 576
7.148.4.3 add_tag() . . . 577
7.148.4.4 lookup_tag() . . . 578
7.148.4.5 remove_tag() . . . 578
7.148.5 Variable Documentation . . . 579
7.148.5.1 DDS_DATATAG_QOS_POLICY_NAME . . . 579
7.149 DATA_WRITER_PROTOCOL . . . 579
7.149.1 Detailed Description . . . 579
7.149.2 Variable Documentation . . . 579
7.149.2.1 DDS_DATAWRITERPROTOCOL_QOS_POLICY_NAME . . . 580
7.150 DATA_WRITER_RESOURCE_LIMITS . . . 580
7.150.1 Detailed Description . . . 580
7.150.2 Enumeration Type Documentation . . . 580
7.150.2.1 DDS_DataWriterResourceLimitsInstanceReplacementKind . . . 581
7.150.3 Variable Documentation . . . 582
7.150.3.1 DDS_DATAWRITERRESOURCELIMITS_QOS_POLICY_NAME . . . 583
7.151 DATA_WRITER_TRANSFER_MODE . . . 583
7.151.1 Detailed Description . . . 583

7.151.2 Variable Documentation . . . 583
7.151.2.1 DDS_DATAWRITERTRANSFERMODE_QOS_POLICY_NAME . . . 583
7.152 DEADLINE . . . 583
7.152.1 Detailed Description . . . 584
7.152.2 Variable Documentation . . . 584
7.152.2.1 DDS_DEADLINE_QOS_POLICY_NAME . . . 584
7.153 DESTINATION_ORDER . . . 584
7.153.1 Detailed Description . . . 585
7.153.2 Enumeration Type Documentation . . . 585
7.153.2.1 DDS_DestinationOrderQosPolicyKind . . . 585
7.153.2.2 DDS_DestinationOrderQosPolicyScopeKind . . . 586
7.153.3 Variable Documentation . . . 586
7.153.3.1 DDS_DESTINATIONORDER_QOS_POLICY_NAME . . . 586
7.154 DISCOVERY . . . 586
7.154.1 Detailed Description . . . 587
7.154.2 Variable Documentation . . . 587
7.154.2.1 DDS_DISCOVERY_QOS_POLICY_NAME . . . 587
7.155 DISCOVERY_CONFIG . . . 587
7.155.1 Detailed Description . . . 589
7.155.2 Macro Definition Documentation . . . 589
7.155.2.1 DDS_DISCOVERYCONFIG_BUILTIN_PLUGIN_MASK_NONE . . . 589
7.155.2.2 DDS_DISCOVERYCONFIG_BUILTIN_PLUGIN_MASK_DEFAULT . . . 589
7.155.2.3 DDS_DISCOVERYCONFIG_BUILTIN_CHANNEL_MASK_NONE . . . 589
7.155.2.4 DDS_DISCOVERYCONFIG_BUILTIN_CHANNEL_MASK_DEFAULT . . . 590
7.155.2.5 DDS_DISCOVERYCONFIG_BUILTIN_CHANNEL_MASK_ALL . . . 590
7.155.3 Typedef Documentation . . . 590
7.155.3.1 DDS_DiscoveryConfigBuiltinPluginKindMask . . . 590
7.155.3.2 DDS_DiscoveryConfigBuiltinChannelKindMask . . . 591
7.155.4 Enumeration Type Documentation . . . 591
7.155.4.1 DDS_DiscoveryConfigBuiltinPluginKind . . . 591
7.155.4.2 DDS_DiscoveryConfigBuiltinChannelKind . . . 592
7.155.4.3 DDS_RemoteParticipantPurgeKind . . . 593
7.155.5 Variable Documentation . . . 594
7.155.5.1 DDS_DISCOVERYCONFIG_QOS_POLICY_NAME . . . 594
7.156 DOMAIN_PARTICIPANT_RESOURCE_LIMITS . . . 594
7.156.1 Detailed Description . . . 595
7.156.2 Enumeration Type Documentation . . . 595
7.156.2.1 DDS_DomainParticipantResourceLimitsIgnoredEntityReplacementKind . . . 595
7.156.3 Variable Documentation . . . 595

7.156.3.1 DDS_AUTO_COUNT . . . 596
7.156.3.2 DDS_DOMAINPARTICIPANTRESOURCELIMITS_QOS_POLICY_NAME . . . 596
7.157 DURABILITY . . . 596
7.157.1 Detailed Description . . . 597
7.157.2 Enumeration Type Documentation . . . 597
7.157.2.1 DDS_PersistentJournalKind . . . 597
7.157.2.2 DDS_PersistentSynchronizationKind . . . 598
7.157.2.3 DDS_DurabilityQosPolicyKind . . . 598
7.157.3 Variable Documentation . . . 599
7.157.3.1 DDS_DURABILITY_QOS_POLICY_NAME . . . 599
7.157.3.2 DDS_AUTO_WRITER_DEPTH . . . 599
7.158 DURABILITY_SERVICE . . . 600
7.158.1 Detailed Description . . . 600
7.158.2 Variable Documentation . . . 600
7.158.2.1 DDS_DURABILITYSERVICE_QOS_POLICY_NAME . . . 600
7.159 ENTITY_FACTORY . . . 600
7.159.1 Detailed Description . . . 601
7.159.2 Variable Documentation . . . 601
7.159.2.1 DDS_ENTITYFACTORY_QOS_POLICY_NAME . . . 601
7.160 ENTITY_NAME . . . 601
7.160.1 Detailed Description . . . 602
7.160.2 Variable Documentation . . . 602
7.160.2.1 DDS_ENTITYNAME_QOS_POLICY_NAME . . . 602
7.161 EVENT . . . 602
7.161.1 Detailed Description . . . 602
7.161.2 Variable Documentation . . . 603
7.161.2.1 DDS_EVENT_QOS_POLICY_NAME . . . 603
7.162 HISTORY . . . 603
7.162.1 Detailed Description . . . 603
7.162.2 Enumeration Type Documentation . . . 603
7.162.2.1 DDS_HistoryQosPolicyKind . . . 603
7.162.3 Variable Documentation . . . 604
7.162.3.1 DDS_HISTORY_QOS_POLICY_NAME . . . 604
7.163 GROUP_DATA . . . 604
7.163.1 Detailed Description . . . 605
7.163.2 Variable Documentation . . . 605
7.163.2.1 DDS_GROUPDATA_QOS_POLICY_NAME . . . 605
7.164 LATENCY_BUDGET . . . 605
7.164.1 Detailed Description . . . 605

7.164.2 Variable Documentation . . . 606
7.164.2.1 DDS_LATENCYBUDGET_QOS_POLICY_NAME . . . 606
7.165 LIFESPAN . . . 606
7.165.1 Detailed Description . . . 606
7.165.2 Variable Documentation . . . 606
7.165.2.1 DDS_LIFESPAN_QOS_POLICY_NAME . . . 606
7.166 LIVELINESS . . . 607
7.166.1 Detailed Description . . . 607
7.166.2 Enumeration Type Documentation . . . 607
7.166.2.1 DDS_LivelinessQosPolicyKind . . . 607
7.166.3 Variable Documentation . . . 608
7.166.3.1 DDS_LIVELINESS_QOS_POLICY_NAME . . . 608
7.167 LOCATORFILTER . . . 608
7.167.1 Detailed Description . . . 609
7.167.2 Variable Documentation . . . 609
7.167.2.1 DDS_LOCATORFILTER_QOS_POLICY_NAME . . . 609
7.168 LOGGING . . . 609
7.168.1 Detailed Description . . . 610
7.169 MONITORING . . . 610
7.169.1 Detailed Description . . . 610
7.169.2 Variable Documentation . . . 611
7.169.2.1 DDS_MONITORING_QOS_POLICY_NAME . . . 611
7.170 MULTICHANNEL . . . 611
7.170.1 Detailed Description . . . 611
7.170.2 Variable Documentation . . . 611
7.170.2.1 DDS_MULTICHANNEL_QOS_POLICY_NAME . . . 612
7.171 OWNERSHIP . . . 612
7.171.1 Detailed Description . . . 612
7.171.2 Enumeration Type Documentation . . . 612
7.171.2.1 DDS_OwnershipQosPolicyKind . . . 612
7.171.3 Variable Documentation . . . 613
7.171.3.1 DDS_OWNERSHIP_QOS_POLICY_NAME . . . 613
7.172 OWNERSHIP_STRENGTH . . . 613
7.172.1 Detailed Description . . . 614
7.172.2 Variable Documentation . . . 614
7.172.2.1 DDS_OWNERSHIPSTRENGTH_QOS_POLICY_NAME . . . 614
7.173 PARTITION . . . 614
7.173.1 Detailed Description . . . 614
7.173.2 Variable Documentation . . . 614

7.173.2.1 DDS_PARTITION_QOS_POLICY_NAME . . . 615
7.174 PRESENTATION . . . 615
7.174.1 Detailed Description . . . 615
7.174.2 Enumeration Type Documentation . . . 615
7.174.2.1 DDS_PresentationQosPolicyAccessScopeKind . . . 615
7.174.3 Variable Documentation . . . 616
7.174.3.1 DDS_PRESENTATION_QOS_POLICY_NAME . . . 616
7.175 PROFILE . . . 616
7.175.1 Detailed Description . . . 617
7.175.2 Variable Documentation . . . 617
7.175.2.1 DDS_PROFILE_QOS_POLICY_NAME . . . 617
7.176 PROPERTY . . . 617
7.176.1 Detailed Description . . . 618
7.176.2 Typedef Documentation . . . 619
7.176.2.1 DDS_PropertyQosPolicy . . . 619
7.176.3 Usage . . . 619
7.176.3.1 Reasons for Using the PropertyQosPolicy . . . 620
7.176.3.2 DDS_PropertyQosPolicyMutability . . . 620
7.176.4 Enumeration Type Documentation . . . 620
7.176.4.1 DDS_PropertyQosPolicyMutability . . . 620
7.176.5 Function Documentation . . . 621
7.176.5.1 DDS_PropertyQosPolicyHelper_get_property_mutability() . . . 621
7.176.5.2 get_number_of_properties() . . . 621
7.176.5.3 assert_property() . . . 622
7.176.5.4 add_property() . . . 622
7.176.5.5 assert_pointer_property() . . . 623
7.176.5.6 add_pointer_property() . . . 624
7.176.5.7 lookup_property() . . . 625
7.176.5.8 remove_property() . . . 625
7.176.5.9 get_properties() . . . 626
7.176.6 Variable Documentation . . . 626
7.176.6.1 DDS_PROPERTY_QOS_POLICY_NAME . . . 627
7.177 PUBLISH_MODE . . . 627
7.177.1 Detailed Description . . . 627
7.177.2 Macro Definition Documentation . . . 628
7.177.2.1 DDS_PUBLICATION_PRIORITY_UNDEFINED . . . 628
7.177.2.2 DDS_PUBLICATION_PRIORITY_AUTOMATIC . . . 628
7.177.3 Enumeration Type Documentation . . . 628
7.177.3.1 DDS_PublishModeQosPolicyKind . . . 628

7.177.4 Variable Documentation . . . 629
7.177.4.1 DDS_PUBLISHMODE_QOS_POLICY_NAME . . . 630
7.178 READER_DATA_LIFECYCLE . . . 630
7.178.1 Detailed Description . . . 630
7.178.2 Variable Documentation . . . 630
7.178.2.1 DDS_READERDATALIFECYCLE_QOS_POLICY_NAME . . . 630
7.179 RECEIVER_POOL . . . 630
7.179.1 Detailed Description . . . 631
7.179.2 Variable Documentation . . . 631
7.179.2.1 DDS_LENGTH_AUTO . . . 631
7.179.2.2 DDS_RECEIVERPOOL_QOS_POLICY_NAME . . . 631
7.180 RELIABILITY . . . 631
7.180.1 Detailed Description . . . 632
7.180.2 Enumeration Type Documentation . . . 632
7.180.2.1 DDS_ReliabilityQosPolicyKind . . . 632
7.180.2.2 DDS_ReliabilityQosPolicyAcknowledgmentModeKind . . . 633
7.180.2.3 DDS_InstanceStateConsistencyKind . . . 634
7.180.3 Variable Documentation . . . 635
7.180.3.1 DDS_RELIABILITY_QOS_POLICY_NAME . . . 635
7.181 RESOURCE_LIMITS . . . 635
7.181.1 Detailed Description . . . 635
7.181.2 Variable Documentation . . . 635
7.181.2.1 DDS_RESOURCELIMITS_QOS_POLICY_NAME . . . 636
7.181.2.2 DDS_LENGTH_UNLIMITED . . . 636
7.182 SERVICE . . . 636
7.182.1 Detailed Description . . . 637
7.182.2 Enumeration Type Documentation . . . 637
7.182.2.1 DDS_ServiceQosPolicyKind . . . 637
7.182.3 Variable Documentation . . . 638
7.182.3.1 DDS_SERVICE_QOS_POLICY_NAME . . . 638
7.183 SYSTEM_RESOURCE_LIMITS . . . 638
7.183.1 Detailed Description . . . 638
7.183.2 Variable Documentation . . . 638
7.183.2.1 DDS_SYSTEMRESOURCELIMITS_QOS_POLICY_NAME . . . 638
7.184 TIME_BASED_FILTER . . . 639
7.184.1 Detailed Description . . . 639
7.184.2 Variable Documentation . . . 639
7.184.2.1 DDS_TIMEBASEDFILTER_QOS_POLICY_NAME . . . 639
7.185 TOPIC_DATA . . . 639

7.185.1 Detailed Description . . . 640
7.185.2 Variable Documentation . . . 640
7.185.2.1 DDS_TOPICDATA_QOS_POLICY_NAME . . . 640
7.186 TOPIC_QUERY_DISPATCH . . . 640
7.186.1 Detailed Description . . . 640
7.186.2 Variable Documentation . . . 641
7.186.2.1 DDS_TOPICQUERYDISPATCH_QOS_POLICY_NAME . . . 641
7.187 TRANSPORT_BUILTIN . . . 641
7.187.1 Detailed Description . . . 642
7.187.2 Macro Definition Documentation . . . 642
7.187.2.1 DDS_TRANSPORTBUILTIN_MASK_NONE . . . 642
7.187.2.2 DDS_TRANSPORTBUILTIN_MASK_DEFAULT . . . 643
7.187.2.3 DDS_TRANSPORTBUILTIN_MASK_ALL . . . 643
7.187.3 Typedef Documentation . . . 643
7.187.3.1 DDS_TransportBuiltinKindMask . . . 643
7.187.4 Enumeration Type Documentation . . . 643
7.187.4.1 DDS_TransportBuiltinKind . . . 643
7.187.5 Variable Documentation . . . 644
7.187.5.1 DDS_TRANSPORTBUILTIN_QOS_POLICY_NAME . . . 644
7.187.5.2 DDS_TRANSPORTBUILTIN_SHMEM_ALIAS . . . 644
7.187.5.3 DDS_TRANSPORTBUILTIN_UDPv4_ALIAS . . . 644
7.187.5.4 DDS_TRANSPORTBUILTIN_UDPv4_WAN_ALIAS . . . 644
7.187.5.5 DDS_TRANSPORTBUILTIN_UDPv6_ALIAS . . . 645
7.188 TRANSPORT_MULTICAST . . . 645
7.188.1 Detailed Description . . . 645
7.188.2 Enumeration Type Documentation . . . 646
7.188.2.1 DDS_TransportMulticastQosPolicyKind . . . 646
7.188.3 Variable Documentation . . . 646
7.188.3.1 DDS_TRANSPORTMULTICAST_QOS_POLICY_NAME . . . 646
7.189 TRANSPORT_MULTICAST_MAPPING . . . 646
7.189.1 Detailed Description . . . 647
7.189.2 Variable Documentation . . . 647
7.189.2.1 DDS_TRANSPORTMULTICASTMAPPING_QOS_POLICY_NAME . . . 647
7.190 TRANSPORT_PRIORITY . . . 647
7.190.1 Detailed Description . . . 647
7.190.2 Variable Documentation . . . 648
7.190.2.1 DDS_TRANSPORTPRIORITY_QOS_POLICY_NAME . . . 648
7.191 TRANSPORT_SELECTION . . . 648
7.191.1 Detailed Description . . . 648

7.191.2 Variable Documentation . . . 648
7.191.2.1 DDS_TRANSPORTSELECTION_QOS_POLICY_NAME . . . 648
7.192 TRANSPORT_UNICAST . . . 649
7.192.1 Detailed Description . . . 649
7.192.2 Variable Documentation . . . 649
7.192.2.1 DDS_TRANSPORTUNICAST_QOS_POLICY_NAME . . . 649
7.193 TYPE_CONSISTENCY_ENFORCEMENT . . . 649
7.193.1 Detailed Description . . . 650
7.193.2 Enumeration Type Documentation . . . 650
7.193.2.1 DDS_TypeConsistencyKind . . . 650
7.193.3 Variable Documentation . . . 651
7.193.3.1 DDS_TYPE_CONSISTENCY_ENFORCEMENT_QOS_POLICY_NAME . . . 651
7.194 TYPESUPPORT . . . 651
7.194.1 Detailed Description . . . 652
7.194.2 Enumeration Type Documentation . . . 652
7.194.2.1 DDS_CdrPaddingKind . . . 652
7.194.3 Variable Documentation . . . 653
7.194.3.1 DDS_TYPESUPPORT_QOS_POLICY_NAME . . . 653
7.195 USER_DATA . . . 653
7.195.1 Detailed Description . . . 653
7.195.2 Variable Documentation . . . 653
7.195.2.1 DDS_USERDATA_QOS_POLICY_NAME . . . 653
7.196 WRITER_DATA_LIFECYCLE . . . 654
7.196.1 Detailed Description . . . 654
7.196.2 Variable Documentation . . . 654
7.196.2.1 DDS_WRITERDATALIFECYCLE_QOS_POLICY_NAME . . . 654
7.197 WIRE_PROTOCOL . . . 654
7.197.1 Detailed Description . . . 655
7.197.2 Macro Definition Documentation . . . 655
7.197.2.1 DDS_RTPS_RESERVED_PORT_MASK_DEFAULT . . . 656
7.197.2.2 DDS_RTPS_RESERVED_PORT_MASK_NONE . . . 656
7.197.2.3 DDS_RTPS_RESERVED_PORT_MASK_ALL . . . 657
7.197.3 Typedef Documentation . . . 657
7.197.3.1 DDS_RtpsReservedPortKindMask . . . 657
7.197.4 Enumeration Type Documentation . . . 657
7.197.4.1 DDS_RtpsReservedPortKind . . . 657
7.197.4.2 DDS_WireProtocolQosPolicyAutoKind . . . 658
7.197.5 Variable Documentation . . . 658
7.197.5.1 DDS_RTI_BACKWARDS_COMPATIBLE_RTPS_WELL_KNOWN_PORTS . . . 658

7.197.5.2 DDS_INTEROPERABLE_RTPS_WELL_KNOWN_PORTS . . . 659
7.197.5.3 DDS_WIREPROTOCOL_QOS_POLICY_NAME . . . 660
7.198 Extended Qos Support . . . 660
7.198.1 Detailed Description . . . 660
7.199 Unicast Settings . . . 660
7.199.1 Detailed Description . . . 660
7.200 Multicast Settings . . . 661
7.200.1 Detailed Description . . . 661
7.201 Multicast Mapping . . . 661
7.201.1 Detailed Description . . . 661
7.202 NDDS_DISCOVERY_PEERS . . . 662
7.202.1 Peer Descriptor Format . . . 663
7.202.1.1 Locator Format . . . 663
7.202.1.2 Address Format . . . 664
7.202.2 NDDS_DISCOVERY_PEERS Environment Variable Format . . . 664
7.202.3 NDDS_DISCOVERY_PEERS File Format . . . 666
7.202.4 NDDS_DISCOVERY_PEERS Precedence . . . 667
7.202.5 NDDS_DISCOVERY_PEERS Default Value . . . 667
7.202.6 Builtin Transport Class Names . . . 668
7.202.7 NDDS_DISCOVERY_PEERS and Local Host Communication . . . 668
7.203 Entity Support . . . 669
7.203.1 Detailed Description . . . 669
7.204 Conditions and WaitSets . . . 670
7.204.1 Detailed Description . . . 670
7.204.2 Macro Definition Documentation . . . 670
7.204.2.1 DDS_WaitSetProperty_t_INITIALIZER . . . 671
7.205 Cookie . . . 671
7.205.1 Detailed Description . . . 671
7.205.2 Function Documentation . . . 671
7.205.2.1 to_pointer() . . . 672
7.206 Sample Flags . . . 672
7.206.1 Detailed Description . . . 672
7.206.2 Typedef Documentation . . . 673
7.206.2.1 DDS_SampleFlagBits . . . 673
7.206.2.2 DDS_SampleFlag . . . 673
7.206.3 Enumeration Type Documentation . . . 673
7.206.3.1 DDS_SampleFlagBits . . . 673
7.207 WriteParams . . . 674
7.207.1 Detailed Description . . . 675

7.207.2 Function Documentation . . . 675
7.207.2.1 DDS_SampleIdentity_equals() . . . 675
7.207.2.2 DDS_WriteParams_reset() . . . 676
7.207.3 Variable Documentation . . . 676
7.207.3.1 writer_guid . . . 676
7.207.3.2 sequence_number . . . 676
7.207.3.3 DDS_AUTO_SAMPLE_IDENTITY . . . 676
7.207.3.4 DDS_UNKNOWN_SAMPLE_IDENTITY . . . 677
7.207.3.5 DDS_WRITEPARAMS_DEFAULT . . . 677
7.208 Heap Support in C . . . 677
7.208.1 Detailed Description . . . 677
7.208.2 Function Documentation . . . 678
7.208.2.1 DDS_Heap_calloc() . . . 678
7.208.2.2 DDS_Heap_malloc() . . . 678
7.208.2.3 DDS_Heap_free() . . . 679
7.209 Builtin Qos Profiles . . . 679
7.209.1 Detailed Description . . . 683
7.209.2 Variable Documentation . . . 685
7.209.2.1 DDS_BUILTIN_QOS_LIB . . . 685
7.209.2.2 DDS_PROFILE_BASELINE_ROOT . . . 685
7.209.2.3 DDS_PROFILE_BASELINE . . . 685
7.209.2.4 DDS_PROFILE_BASELINE_5_0_0 . . . 685
7.209.2.5 DDS_PROFILE_BASELINE_5_1_0 . . . 686
7.209.2.6 DDS_PROFILE_BASELINE_5_2_0 . . . 686
7.209.2.7 DDS_PROFILE_BASELINE_5_3_0 . . . 686
7.209.2.8 DDS_PROFILE_BASELINE_6_0_0 . . . 686
7.209.2.9 DDS_PROFILE_BASELINE_6_1_0 . . . 686
7.209.2.10 DDS_PROFILE_BASELINE_7_0_0 . . . 687
7.209.2.11 DDS_PROFILE_BASELINE_7_1_0 . . . 687
7.209.2.12 DDS_PROFILE_GENERIC_COMMON . . . 687
7.209.2.13 DDS_PROFILE_GENERIC_MONITORING_COMMON . . . 687
7.209.2.14 DDS_PROFILE_GENERIC_CONNEXT_MICRO_COMPATIBILITY . . . 688
7.209.2.15 DDS_PROFILE_GENERIC_CONNEXT_MICRO_COMPATIBILITY_2_4_9 . . . 688
7.209.2.16 DDS_PROFILE_GENERIC_CONNEXT_MICRO_COMPATIBILITY_2_4_3 . . . 688
7.209.2.17 DDS_PROFILE_GENERIC_OTHER_DDS_VENDOR_COMPATIBILITY . . . 688
7.209.2.18 DDS_PROFILE_GENERIC_510_TRANSPORT_COMPATIBILITY . . . 689
7.209.2.19 DDS_BUILTIN_QOS_LIB_EXP . . . 689
7.209.2.20 DDS_PROFILE_GENERIC_STRICT_RELIABLE . . . 689
7.209.2.21 DDS_PROFILE_GENERIC_KEEP_LAST_RELIABLE . . . 689

7.209.2.22 DDS_PROFILE_GENERIC_BEST_EFFORT 690
7.209.2.23 DDS_PROFILE_GENERIC_STRICT_RELIABLE_HIGH_THROUGHPUT 690
7.209.2.24 DDS_PROFILE_GENERIC_STRICT_RELIABLE_LOW_LATENCY 690
7.209.2.25 DDS_PROFILE_GENERIC_PARTICIPANT_LARGE_DATA 691
7.209.2.26 DDS_PROFILE_GENERIC_PARTICIPANT_LARGE_DATA_MONITORING 691
7.209.2.27 DDS_PROFILE_GENERIC_STRICT_RELIABLE_LARGE_DATA 692
7.209.2.28 DDS_PROFILE_GENERIC_KEEP_LAST_RELIABLE_LARGE_DATA 692
7.209.2.29 DDS_PROFILE_GENERIC_STRICT_RELIABLE_LARGE_DATA_FAST_FLOW . . . 692
7.209.2.30 DDS_PROFILE_GENERIC_STRICT_RELIABLE_LARGE_DATA_MEDIUM_FLOW . 693
7.209.2.31 DDS_PROFILE_GENERIC_STRICT_RELIABLE_LARGE_DATA_SLOW_FLOW . . . 693
7.209.2.32 DDS_PROFILE_GENERIC_KEEP_LAST_RELIABLE_LARGE_DATA_FAST_FLOW . 693
7.209.2.33 DDS_PROFILE_GENERIC_KEEP_LAST_RELIABLE_LARGE_DATA_MEDIUM_FLOW693
7.209.2.34 DDS_PROFILE_GENERIC_KEEP_LAST_RELIABLE_LARGE_DATA_SLOW_FLOW 694
7.209.2.35 DDS_PROFILE_GENERIC_KEEP_LAST_RELIABLE_TRANSIENT_LOCAL 694
7.209.2.36 DDS_PROFILE_GENERIC_KEEP_LAST_RELIABLE_TRANSIENT 694
7.209.2.37 DDS_PROFILE_GENERIC_KEEP_LAST_RELIABLE_PERSISTENT 694
7.209.2.38 DDS_PROFILE_GENERIC_AUTO_TUNING 695
7.209.2.39 DDS_PROFILE_GENERIC_MINIMAL_MEMORY_FOOTPRINT 695
7.209.2.40 DDS_PROFILE_GENERIC_SECURITY 695
7.209.2.41 DDS_PROFILE_GENERIC_MONITORING2 696
7.209.2.42 DDS_PROFILE_GENERIC_MONITORING2_WAN 697
7.209.2.43 DDS_PROFILE_PATTERN_RPC 697
7.209.2.44 DDS_PROFILE_PATTERN_PERIODIC_DATA 697
7.209.2.45 DDS_PROFILE_PATTERN_STREAMING 698
7.209.2.46 DDS_PROFILE_PATTERN_RELIABLE_STREAMING 698
7.209.2.47 DDS_PROFILE_PATTERN_EVENT 698
7.209.2.48 DDS_PROFILE_PATTERN_ALARM_EVENT 699
7.209.2.49 DDS_PROFILE_PATTERN_STATUS 699
7.209.2.50 DDS_PROFILE_PATTERN_ALARM_STATUS 699
7.209.2.51 DDS_PROFILE_PATTERN_LAST_VALUE_CACHE 700
7.209.2.52 DDS_BUILTIN_QOS_SNIPPET_LIB 700
7.209.2.53 DDS_SNIPPET_OPTIMIZATION_RELIABILITY_PROTOCOL_COMMON 700
7.209.2.54 DDS_SNIPPET_OPTIMIZATION_RELIABILITY_PROTOCOL_KEEP_ALL 701
7.209.2.55 DDS_SNIPPET_OPTIMIZATION_RELIABILITY_PROTOCOL_KEEP_LAST 701
7.209.2.56 DDS_SNIPPET_OPTIMIZATION_RELIABILITY_PROTOCOL_HIGH_RATE 702
7.209.2.57 DDS_SNIPPET_OPTIMIZATION_RELIABILITY_PROTOCOL_LOW_LATENCY . . . 702
7.209.2.58 DDS_SNIPPET_OPTIMIZATION_RELIABILITY_PROTOCOL_LARGE_DATA 703
7.209.2.59 DDS_SNIPPET_OPTIMIZATION_DATACACHE_LARGE_DATA_DYNAMICMEMALLOC703
7.209.2.60 DDS_SNIPPET_OPTIMIZATION_DISCOVERY_COMMON 704

7.209.2.61 DDS_SNIPPET_OPTIMIZATION_DISCOVERY_PARTICIPANT_COMPACT 704
7.209.2.62 DDS_SNIPPET_OPTIMIZATION_DISCOVERY_ENDPOINT_FAST 705
7.209.2.63 DDS_SNIPPET_OPTIMIZATION_TRANSPORT_LARGE_BUFFERS 705
7.209.2.64 DDS_SNIPPET_QOS_POLICY_RELIABILITY_RELIABLE 705
7.209.2.65 DDS_SNIPPET_QOS_POLICY_RELIABILITY_BEST_EFFORT 706
7.209.2.66 DDS_SNIPPET_QOS_POLICY_HISTORY_KEEP_LAST_1 706
7.209.2.67 DDS_SNIPPET_QOS_POLICY_HISTORY_KEEP_ALL 706
7.209.2.68 DDS_SNIPPET_QOS_POLICY_PUBLISH_MODE_ASYNCHRONOUS 707
7.209.2.69 DDS_SNIPPET_QOS_POLICY_DURABILITY_TRANSIENT_LOCAL 707
7.209.2.70 DDS_SNIPPET_QOS_POLICY_DURABILITY_TRANSIENT 708
7.209.2.71 DDS_SNIPPET_QOS_POLICY_DURABILITY_PERSISTENT 708
7.209.2.72 DDS_SNIPPET_QOS_POLICY_BATCHING_ENABLE 709
7.209.2.73 DDS_SNIPPET_FEATURE_FLOW_CONTROLLER_838MBPS 709
7.209.2.74 DDS_SNIPPET_FEATURE_FLOW_CONTROLLER_209MBPS 709
7.209.2.75 DDS_SNIPPET_FEATURE_FLOW_CONTROLLER_52MBPS 710
7.209.2.76 DDS_SNIPPET_FEATURE_AUTO_TUNING_ENABLE 710
7.209.2.77 DDS_SNIPPET_FEATURE_MONITORING_ENABLE 711
7.209.2.78 DDS_SNIPPET_FEATURE_MONITORING2_ENABLE 711
7.209.2.79 DDS_SNIPPET_FEATURE_SECURITY_ENABLE 711
7.209.2.80 DDS_SNIPPET_FEATURE_TOPIC_QUERY_ENABLE 712
7.209.2.81 DDS_SNIPPET_FEATURE_TYPE_SUPPORT_V1_ENABLE 712
7.209.2.82 DDS_SNIPPET_FEATURE_TYPE_SUPPORT_V1_DISABLE 712
7.209.2.83 DDS_SNIPPET_FEATURE_TYPE_SUPPORT_V2_ENABLE 713
7.209.2.84 DDS_SNIPPET_FEATURE_TYPE_SUPPORT_V2_DISABLE 713
7.209.2.85 DDS_SNIPPET_TRANSPORT_TCP_LAN_CLIENT 713
7.209.2.86 DDS_SNIPPET_TRANSPORT_TCP_WAN_SYMMETRIC_CLIENT 714
7.209.2.87 DDS_SNIPPET_TRANSPORT_TCP_WAN_ASYMMETRIC_SERVER 714
7.209.2.88 DDS_SNIPPET_TRANSPORT_TCP_WAN_ASYMMETRIC_CLIENT 714
7.209.2.89 DDS_SNIPPET_TRANSPORT_UDP_AVOID_IP_FRAGMENTATION 715
7.209.2.90 DDS_SNIPPET_TRANSPORT_UDP_WAN 715
7.209.2.91 DDS_SNIPPET_COMPATIBILITY_CONNEXT_MICRO_VERSION_2_4_3 715
7.209.2.92 DDS_SNIPPET_COMPATIBILITY_OTHER_DDS_VENDORS_ENABLE 716
7.209.2.93 DDS_SNIPPET_5_1_0_TRANSPORT_ENABLE 716
7.210 DomainParticipantConfigParams 716
7.210.1 Detailed Description 717
7.210.2 Macro Definition Documentation 717
7.210.2.1 DDS_DomainParticipantConfigParams_t_INITIALIZER 717
7.210.3 Variable Documentation 717
7.210.3.1 DDS_DOMAIN_ID_USE_XML_CONFIG 717

7.210.3.2 DDS_ENTITY_NAME_USE_XML_CONFIG . . . 717
7.210.3.3 DDS_QOS_ELEMENT_NAME_USE_XML_CONFIG . . . 718
7.211 User-managed Threads . . . 718
7.211.1 Detailed Description . . . 718
7.211.2 Typedef Documentation . . . 718
7.211.2.1 DDSThreadFactory_OnSpawnedFunction . . . 718
7.212 Monitoring Library 2.0 . . . 719
7.212.1 Detailed Description . . . 719
7.212.2 Function Documentation . . . 720
7.212.2.1 RTI_Monitoring_initialize() . . . 720
7.212.3 Variable Documentation . . . 720
7.212.3.1 RTI_MONITORING_PERIODIC_TOPIC_NAME . . . 720
7.212.3.2 RTI_MONITORING_EVENT_TOPIC_NAME . . . 721
7.212.3.3 RTI_MONITORING_LOGGING_TOPIC_NAME . . . 721
7.213 Version . . . 721
7.213.1 Detailed Description . . . 721
7.214 Logging . . . 722
7.214.1 Detailed Description . . . 723
7.214.2 Enumeration Type Documentation . . . 723
7.214.2.1 NDDS_Config_LogVerbosity . . . 723
7.214.2.2 NDDS_Config_LogLevel . . . 724
7.214.2.3 NDDS_Config_SyslogLevel . . . 725
7.214.2.4 NDDS_Config_LogCategory . . . 725
7.214.2.5 NDDS_Config_LogPrintFormat . . . 726
7.214.2.6 NDDS_Config_LogFacility . . . 727
7.214.2.7 NDDS_Config_SyslogVerbosity . . . 727
7.215 Activity Context . . . 729
7.215.1 Detailed Description . . . 730
7.215.2 Macro Definition Documentation . . . 730
7.215.2.1 NDDS_CONFIG_ACTIVITY_CONTEXT_ATTRIBUTE_MASK_DEFAULT . . . 731
7.215.2.2 NDDS_CONFIG_ACTIVITY_CONTEXT_ATTRIBUTE_MASK_NONE . . . 731
7.215.2.3 NDDS_CONFIG_ACTIVITY_CONTEXT_ATTRIBUTE_MASK_ALL . . . 731
7.215.3 Typedef Documentation . . . 731
7.215.3.1 NDDS_Config_ActivityContextAttributeKindMask . . . 731
7.215.4 Enumeration Type Documentation . . . 731
7.215.4.1 NDDS_Config_ActivityContextAttributeKind . . . 731
7.215.5 Function Documentation . . . 733
7.215.5.1 set_attribute_mask() . . . 734
7.216 Compliance Configuration . . . 734

7.216.1 Detailed Description . . . 735
7.216.2 Macro Definition Documentation . . . 735
7.216.2.1 NDDS_CONFIG_XTYPES_COMPLIANCE_MASK_VENDOR . . . 735
7.216.2.2 NDDS_CONFIG_XTYPES_COMPLIANCE_MASK_DEFAULT . . . 736
7.216.2.3 NDDS_CONFIG_RPC_COMPLIANCE_MASK_VENDOR . . . 736
7.216.2.4 NDDS_CONFIG_RPC_COMPLIANCE_MASK_DEFAULT . . . 736
7.216.3 Typedef Documentation . . . 736
7.216.3.1 NDDS_Config_XTypesComplianceMask . . . 737
7.216.3.2 NDDS_Config_RpcComplianceMask . . . 737
7.216.4 Enumeration Type Documentation . . . 738
7.216.4.1 NDDS_Config_XTypesComplianceMaskBits . . . 738
7.216.4.2 NDDS_Config_RpcComplianceMaskBits . . . 745
7.217 Heap Monitoring . . . 746
7.217.1 Detailed Description . . . 746
7.217.2 Enumeration Type Documentation . . . 746
7.217.2.1 NDDS_Utility_HeapMonitoringSnapshotContentFormat . . . 747
7.218 Network Capture . . . 747
7.218.1 Detailed Description . . . 748
7.218.2 Capturing . . . 749
7.218.3 Macro Definition Documentation . . . 750
7.218.3.1 NDDS_UTILITY_NETWORK_CAPTURE_CONTENT_MASK_DEFAULT . . . 750
7.218.3.2 NDDS_UTILITY_NETWORK_CAPTURE_CONTENT_MASK_NONE . . . 750
7.218.3.3 NDDS_UTILITY_NETWORK_CAPTURE_CONTENT_MASK_ALL . . . 750
7.218.3.4 NDDS_UTILITY_NETWORK_CAPTURE_TRAFFIC_MASK_DEFAULT . . . 750
7.218.3.5 NDDS_UTILITY_NETWORK_CAPTURE_TRAFFIC_MASK_NONE . . . 751
7.218.3.6 NDDS_UTILITY_NETWORK_CAPTURE_TRAFFIC_MASK_ALL . . . 751
7.218.4 Typedef Documentation . . . 751
7.218.4.1 NDDS_Utility_NetworkCaptureContentKindMask . . . 751
7.218.4.2 NDDS_Utility_NetworkCaptureTrafficKindMask . . . 751
7.218.4.3 NDDS_Utility_NetworkCaptureParams_t . . . 752
7.218.5 Enumeration Type Documentation . . . 752
7.218.5.1 NDDS_Utility_NetworkCaptureContentKind . . . 752
7.218.5.2 NDDS_Utility_NetworkCaptureTrafficKind . . . 752
7.218.6 Variable Documentation . . . 753
7.218.6.1 NDDS_UTILITY_NETWORK_CAPTURE_PARAMETERS_DEFAULT . . . 753
7.219 Other Utilities . . . 753
7.219.1 Detailed Description . . . 753
7.220 Octet Buffer Support . . . 753
7.220.1 Detailed Description . . . 754

7.220.2 Conventions . . . 754
7.220.3 Usage . . . 755
7.220.4 Function Documentation . . . 755
7.220.4.1 DDS_OctetBuffer_alloc() . . . 755
7.220.4.2 DDS_OctetBuffer_dup() . . . 756
7.220.4.3 DDS_OctetBuffer_free() . . . 756
7.221 SampleProcessor . . . 756
7.222 Sequence Support . . . 757
7.222.1 Detailed Description . . . 757
7.223 String Support . . . 757
7.223.1 Detailed Description . . . 758
7.223.2 String Conventions . . . 758
7.223.3 Usage . . . 759
7.223.4 Function Documentation . . . 759
7.223.4.1 DDS_String_alloc() . . . 760
7.223.4.2 DDS_String_dup() . . . 761
7.223.4.3 DDS_String_replace() . . . 761
7.223.4.4 DDS_String_free() . . . 762
7.223.4.5 DDS_Wstring_alloc() . . . 763
7.223.4.6 DDS_Wstring_length() . . . 763
7.223.4.7 DDS_Wstring_copy() . . . 763
7.223.4.8 DDS_Wstring_copy_and_widen() . . . 764
7.223.4.9 DDS_Wstring_dup() . . . 764
7.223.4.10 DDS_Wstring_dup_and_widen() . . . 765
7.223.4.11 DDS_Wstring_free() . . . 765
7.224 FlatData Builders . . . 766
7.224.1 Detailed Description . . . 766
7.224.2 Builder Error Management . . . 767
7.224.3 Function Documentation . . . 768
7.224.3.1 build_data() . . . 768
7.224.3.2 discard_builder() . . . 768
7.225 FlatData Samples . . . 769
7.225.1 Detailed Description . . . 769
7.225.2 Typedef Documentation . . . 770
7.225.2.1 MyFlatFinal . . . 770
7.225.2.2 MyFlatMutable . . . 770
7.225.2.3 MyFlatUnion . . . 771
7.226 FlatData Offsets . . . 771
7.226.1 Detailed Description . . . 772

7.226.2 Offset Error Management . . . 773
7.226.3 Function Documentation . . . 773
7.226.3.1 plain_cast() [1/2] . . . 774
7.226.3.2 plain_cast() [2/2] . . . 776
7.227 FlatData Topic-Types . . . 776
7.227.1 Detailed Description . . . 776
7.227.2 Publishing FlatData . . . 777
7.227.3 Subscribing to FlatData . . . 778

8 Namespace Documentation **779**

8.1 connext Namespace Reference . . . 779
8.1.1 Detailed Description . . . 781
8.2 rti Namespace Reference . . . 781
8.2.1 Detailed Description . . . 781
8.3 rti::flat Namespace Reference . . . 781
8.3.1 Detailed Description . . . 783

9 Class Documentation **785**

9.1 rti::flat::AbstractAlignedList< ElementOffset > Class Template Reference . . . 785
9.1.1 Detailed Description . . . 785
9.1.2 Member Typedef Documentation . . . 786
9.1.2.1 iterator . . . 786
9.1.3 Member Function Documentation . . . 786
9.1.3.1 begin() . . . 786
9.1.3.2 end() . . . 786
9.2 rti::flat::AbstractBuilder Class Reference . . . 787
9.2.1 Detailed Description . . . 787
9.2.2 Constructor & Destructor Documentation . . . 787
9.2.2.1 ~AbstractBuilder() . . . 787
9.2.3 Member Function Documentation . . . 788
9.2.3.1 discard() . . . 788
9.2.3.2 is_nested() . . . 788
9.2.3.3 is_valid() . . . 788
9.2.3.4 capacity() . . . 789
9.2.3.5 check_failure() . . . 789
9.3 rti::flat::AbstractListBuilder< DHeaderGen > Class Template Reference . . . 789
9.3.1 Detailed Description . . . 790
9.3.2 Member Function Documentation . . . 790
9.3.2.1 element_count() . . . 790
9.4 rti::flat::AbstractPrimitiveList< T > Class Template Reference . . . 790

9.4.1 Detailed Description . . . 790
9.4.2 Member Function Documentation . . . 791
9.4.2.1 get_element() . . . 791
9.4.2.2 set_element() . . . 791
9.5 rti::flat::AbstractSequenceBuilder< DHeaderGen > Class Template Reference . . . 792
9.5.1 Detailed Description . . . 792
9.5.2 Member Function Documentation . . . 792
9.5.2.1 element_count() . . . 793
9.6 rti::flat::AggregationBuilder Class Reference . . . 793
9.6.1 Detailed Description . . . 793
9.7 connext::AlreadyDeletedException Class Reference . . . 793
9.7.1 Detailed Description . . . 794
9.8 connext::BadParameterException Class Reference . . . 794
9.8.1 Detailed Description . . . 794
9.9 DDS_AcknowledgmentInfo Struct Reference . . . 794
9.9.1 Detailed Description . . . 795
9.9.2 Member Data Documentation . . . 795
9.9.2.1 subscription_handle . . . 795
9.9.2.2 sample_identity . . . 795
9.9.2.3 valid_response_data . . . 795
9.9.2.4 response_data . . . 796
9.10 DDS_AckResponseData_t Struct Reference . . . 796
9.10.1 Detailed Description . . . 796
9.10.2 Member Data Documentation . . . 796
9.10.2.1 value . . . 796
9.11 DDS_AllocationSettings_t Struct Reference . . . 796
9.11.1 Detailed Description . . . 797
9.11.2 Member Data Documentation . . . 797
9.11.2.1 initial_count . . . 797
9.11.2.2 max_count . . . 797
9.11.2.3 incremental_count . . . 798
9.12 DDS_AnnotationParameterValue Struct Reference . . . 798
9.12.1 Detailed Description . . . 798
9.13 DDS_AsynchronousPublisherQosPolicy Struct Reference . . . 798
9.13.1 Detailed Description . . . 799
9.13.2 Usage . . . 799
9.13.3 Member Data Documentation . . . 800
9.13.3.1 disable_asynchronous_write . . . 800
9.13.3.2 thread . . . 800

9.13.3.3 disable_asynchronous_batch . . . 801
9.13.3.4 asynchronous_batch_thread . . . 801
9.13.3.5 disable_topic_query_publication . . . 802
9.13.3.6 topic_query_publication_thread . . . 802
9.14 DDS_AsyncWaitSetProperty_t Struct Reference . . . 802
9.14.1 Detailed Description . . . 803
9.14.2 Member Data Documentation . . . 803
9.14.2.1 waitset_property . . . 803
9.14.2.2 thread_pool_size . . . 804
9.14.2.3 thread_settings . . . 804
9.14.2.4 thread_name_prefix . . . 804
9.14.2.5 wait_timeout . . . 805
9.14.2.6 level . . . 805
9.15 DDS_AvailabilityQosPolicy Struct Reference . . . 805
9.15.1 Detailed Description . . . 806
9.15.2 Usage . . . 806
9.15.3 Consistency . . . 807
9.15.4 Member Data Documentation . . . 808
9.15.4.1 enable_required_subscriptions . . . 808
9.15.4.2 max_data_availability_waiting_time . . . 808
9.15.4.3 max_endpoint_availability_waiting_time . . . 809
9.15.4.4 required_matched_endpoint_groups . . . 809
9.16 DDS_BatchQosPolicy Struct Reference . . . 810
9.16.1 Detailed Description . . . 810
9.16.2 Member Data Documentation . . . 810
9.16.2.1 enable . . . 811
9.16.2.2 max_data_bytes . . . 811
9.16.3 Consistency . . . 811
9.16.3.1 max_samples . . . 812
9.16.4 Consistency . . . 812
9.16.4.1 max_flush_delay . . . 812
9.16.5 Consistency . . . 812
9.16.5.1 source_timestamp_resolution . . . 813
9.16.6 Consistency . . . 813
9.16.6.1 thread_safe_write . . . 813
9.16.7 Consistency . . . 814
9.17 DDS_BooleanSeq Struct Reference . . . 814
9.17.1 Detailed Description . . . 814
9.18 DDS_BuiltinTopicKey_t Struct Reference . . . 814

9.18.1 Detailed Description . . . 815
9.19 DDS_BuiltinTopicReaderResourceLimits_t Struct Reference . . . 815
9.19.1 Detailed Description . . . 816
9.19.2 Member Data Documentation . . . 816
9.19.2.1 initial_samples . . . 816
9.19.2.2 max_samples . . . 817
9.19.2.3 initial_infos . . . 817
9.19.2.4 max_infos . . . 817
9.19.2.5 initial_outstanding_reads . . . 818
9.19.2.6 max_outstanding_reads . . . 818
9.19.2.7 max_samples_per_read . . . 818
9.19.2.8 disable_fragmentation_support . . . 819
9.19.2.9 max_fragmented_samples . . . 819
9.19.2.10 initial_fragmented_samples . . . 820
9.19.2.11 max_fragmented_samples_per_remote_writer . . . 820
9.19.2.12 max_fragments_per_sample . . . 821
9.19.2.13 dynamically_allocate_fragmented_samples . . . 821
9.20 DDS_ChannelSettings_t Struct Reference . . . 822
9.20.1 Detailed Description . . . 822
9.20.2 Member Data Documentation . . . 822
9.20.2.1 multicast_settings . . . 822
9.20.2.2 filter_expression . . . 823
9.20.2.3 priority . . . 823
9.21 DDS_ChannelSettingsSeq Struct Reference . . . 824
9.21.1 Detailed Description . . . 824
9.22 DDS_CharSeq Struct Reference . . . 824
9.22.1 Detailed Description . . . 824
9.23 DDS_CoherentSetInfo_t Struct Reference . . . 824
9.23.1 Detailed Description . . . 825
9.23.2 Member Data Documentation . . . 825
9.23.2.1 group_guid . . . 825
9.23.2.2 coherent_set_sequence_number . . . 825
9.23.2.3 group_coherent_set_sequence_number . . . 826
9.23.2.4 incomplete_coherent_set . . . 826
9.24 DDS_CompressionSettings_t Struct Reference . . . 826
9.24.1 Detailed Description . . . 826
9.24.2 Member Data Documentation . . . 826
9.24.2.1 compression_ids . . . 827
9.24.2.2 writer_compression_level . . . 827

9.24.2.3 writer_compression_threshold . . . 828
9.25 DDS_ContentFilterProperty_t Struct Reference . . . 828
9.25.1 Detailed Description . . . 829
9.25.2 Member Data Documentation . . . 829
9.25.2.1 content_filter_topic_name . . . 829
9.25.2.2 related_topic_name . . . 829
9.25.2.3 filter_class_name . . . 829
9.25.2.4 filter_expression . . . 829
9.25.2.5 expression_parameters . . . 830
9.26 DDS_Cookie_t Struct Reference . . . 830
9.26.1 Detailed Description . . . 830
9.26.2 Member Data Documentation . . . 830
9.26.2.1 value . . . 830
9.27 DDS_CookieSeq Struct Reference . . . 830
9.27.1 Detailed Description . . . 831
9.28 DDS_DatabaseQosPolicy Struct Reference . . . 831
9.28.1 Detailed Description . . . 831
9.28.2 Member Data Documentation . . . 832
9.28.2.1 thread . . . 832
9.28.2.2 shutdown_timeout . . . 833
9.28.2.3 cleanup_period . . . 833
9.28.2.4 shutdown_cleanup_period . . . 833
9.28.2.5 initial_records . . . 834
9.28.2.6 max_skiplist_level . . . 834
9.28.2.7 max_weak_references . . . 835
9.28.2.8 initial_weak_references . . . 835
9.29 DDS_DataReaderCacheStatus Struct Reference . . . 836
9.29.1 Detailed Description . . . 837
9.29.2 Member Data Documentation . . . 837
9.29.2.1 sample_count_peak . . . 837
9.29.2.2 sample_count . . . 837
9.29.2.3 old_source_timestamp_dropped_sample_count . . . 838
9.29.2.4 tolerance_source_timestamp_dropped_sample_count . . . 838
9.29.2.5 ownership_dropped_sample_count . . . 838
9.29.2.6 content_filter_dropped_sample_count . . . 838
9.29.2.7 time_based_filter_dropped_sample_count . . . 839
9.29.2.8 expired_dropped_sample_count . . . 839
9.29.2.9 virtual_duplicate_dropped_sample_count . . . 839
9.29.2.10 replaced_dropped_sample_count . . . 840

9.29.2.11 writer_removed_batch_sample_dropped_sample_count . . . 840
9.29.2.12 total_samples_dropped_by_instance_replacement . . . 840
9.29.2.13 alive_instance_count . . . 840
9.29.2.14 alive_instance_count_peak . . . 841
9.29.2.15 no_writers_instance_count . . . 841
9.29.2.16 no_writers_instance_count_peak . . . 841
9.29.2.17 disposed_instance_count . . . 841
9.29.2.18 disposed_instance_count_peak . . . 842
9.29.2.19 detached_instance_count . . . 842
9.29.2.20 detached_instance_count_peak . . . 842
9.29.2.21 compressed_sample_count . . . 842
9.30 DDS_DataReaderProtocolQosPolicy Struct Reference . . . 843
9.30.1 Detailed Description . . . 843
9.30.2 Member Data Documentation . . . 844
9.30.2.1 virtual_guid . . . 844
9.30.2.2 rtps_object_id . . . 844
9.30.2.3 expects_inline_qos . . . 845
9.30.2.4 disable_positive_acks . . . 845
9.30.2.5 propagate_dispose_of_unregistered_instances . . . 846
9.30.2.6 propagate_unregister_of_disposed_instances . . . 846
9.30.2.7 rtps_reliable_reader . . . 847
9.31 DDS_DataReaderProtocolStatus Struct Reference . . . 847
9.31.1 Detailed Description . . . 849
9.31.2 Member Data Documentation . . . 849
9.31.2.1 received_sample_count . . . 849
9.31.2.2 received_sample_count_change . . . 850
9.31.2.3 received_sample_bytes . . . 850
9.31.2.4 received_sample_bytes_change . . . 851
9.31.2.5 duplicate_sample_count . . . 851
9.31.2.6 duplicate_sample_count_change . . . 851
9.31.2.7 duplicate_sample_bytes . . . 851
9.31.2.8 duplicate_sample_bytes_change . . . 851
9.31.2.9 filtered_sample_count . . . 852
9.31.2.10 filtered_sample_count_change . . . 852
9.31.2.11 filtered_sample_bytes . . . 852
9.31.2.12 filtered_sample_bytes_change . . . 852
9.31.2.13 received_heartbeat_count . . . 852
9.31.2.14 received_heartbeat_count_change . . . 853
9.31.2.15 received_heartbeat_bytes . . . 853

9.31.2.16 received_heartbeat_bytes_change . . . 853
9.31.2.17 sent_ack_count . . . 853
9.31.2.18 sent_ack_count_change . . . 853
9.31.2.19 sent_ack_bytes . . . 854
9.31.2.20 sent_ack_bytes_change . . . 854
9.31.2.21 sent_nack_count . . . 854
9.31.2.22 sent_nack_count_change . . . 854
9.31.2.23 sent_nack_bytes . . . 854
9.31.2.24 sent_nack_bytes_change . . . 854
9.31.2.25 received_gap_count . . . 855
9.31.2.26 received_gap_count_change . . . 855
9.31.2.27 received_gap_bytes . . . 855
9.31.2.28 received_gap_bytes_change . . . 855
9.31.2.29 rejected_sample_count . . . 855
9.31.2.30 rejected_sample_count_change . . . 856
9.31.2.31 first_available_sample_sequence_number . . . 856
9.31.2.32 last_available_sample_sequence_number . . . 856
9.31.2.33 last_committed_sample_sequence_number . . . 856
9.31.2.34 uncommitted_sample_count . . . 856
9.31.2.35 out_of_range_rejected_sample_count . . . 857
9.31.2.36 received_fragment_count . . . 857
9.31.2.37 dropped_fragment_count . . . 857
9.31.2.38 reassembled_sample_count . . . 857
9.31.2.39 sent_nack_fragment_count . . . 857
9.31.2.40 sent_nack_fragment_bytes . . . 858
9.32 DDS_DataReaderQos Struct Reference . . . 858
9.32.1 Detailed Description . . . 860
9.32.2 Member Function Documentation . . . 860
9.32.2.1 operator==() . . . 861
9.32.2.2 operator"!=() . . . 861
9.32.3 Member Data Documentation . . . 861
9.32.3.1 durability . . . 861
9.32.3.2 deadline . . . 861
9.32.3.3 latency_budget . . . 862
9.32.3.4 liveliness . . . 862
9.32.3.5 reliability . . . 862
9.32.3.6 destination_order . . . 862
9.32.3.7 history . . . 862
9.32.3.8 resource_limits . . . 862

9.32.3.9 user_data . . . 863
9.32.3.10 ownership . . . 863
9.32.3.11 time_based_filter . . . 863
9.32.3.12 reader_data_lifecycle . . . 863
9.32.3.13 representation . . . 863
9.32.3.14 type_consistency . . . 863
9.32.3.15 data_tags . . . 864
9.32.3.16 reader_resource_limits . . . 864
9.32.3.17 protocol . . . 864
9.32.3.18 transport_selection . . . 864
9.32.3.19 unicast . . . 864
9.32.3.20 multicast . . . 865
9.32.3.21 property . . . 865
9.32.3.22 service . . . 865
9.32.3.23 availability . . . 865
9.32.3.24 subscription_name . . . 865
9.32.3.25 transport_priority . . . 866
9.32.3.26 type_support . . . 866
9.33 DDS_DataReaderResourceLimitsInstanceReplacementSettings Struct Reference . . . 866
9.33.1 Detailed Description . . . 866
9.33.2 Member Data Documentation . . . 867
9.33.2.1 alive_instance_removal . . . 867
9.33.2.2 disposed_instance_removal . . . 867
9.33.2.3 no_writers_instance_removal . . . 867
9.34 DDS_DataReaderResourceLimitsQosPolicy Struct Reference . . . 867
9.34.1 Detailed Description . . . 869
9.34.2 Member Data Documentation . . . 870
9.34.2.1 max_remote_writers . . . 870
9.34.2.2 max_remote_writers_per_instance . . . 870
9.34.2.3 max_samples_per_remote_writer . . . 871
9.34.2.4 max_infos . . . 871
9.34.2.5 initial_remote_writers . . . 872
9.34.2.6 initial_remote_writers_per_instance . . . 872
9.34.2.7 initial_infos . . . 873
9.34.2.8 initial_outstanding_reads . . . 873
9.34.2.9 max_outstanding_reads . . . 873
9.34.2.10 max_samples_per_read . . . 874
9.34.2.11 disable_fragmentation_support . . . 874
9.34.2.12 max_fragmented_samples . . . 875

9.34.2.13 initial_fragmented_samples . . . 875
9.34.2.14 max_fragmented_samples_per_remote_writer . . . 876
9.34.2.15 max_fragments_per_sample . . . 876
9.34.2.16 dynamically_allocate_fragmented_samples . . . 877
9.34.2.17 max_total_instances . . . 877
9.34.2.18 max_remote_virtual_writers . . . 878
9.34.2.19 initial_remote_virtual_writers . . . 879
9.34.2.20 max_remote_virtual_writers_per_instance . . . 879
9.34.2.21 initial_remote_virtual_writers_per_instance . . . 880
9.34.2.22 max_remote_writers_per_sample . . . 880
9.34.2.23 max_query_condition_filters . . . 881
9.34.2.24 max_app_ack_response_length . . . 881
9.34.2.25 keep_minimum_state_for_instances . . . 882
9.34.2.26 initial_topic_queries . . . 882
9.34.2.27 max_topic_queries . . . 883
9.34.2.28 shmem_ref_transfer_mode_attached_segment_allocation . . . 883
9.34.2.29 instance_replacement . . . 884
9.34.2.30 autopurge_remote_not_alive_writer_delay . . . 885
9.34.2.31 lazily_allocate_batch_resources . . . 886
9.35 DDS_DataRepresentationIdSeq Struct Reference . . . 886
9.35.1 Detailed Description . . . 886
9.36 DDS_DataRepresentationQosPolicy Struct Reference . . . 886
9.36.1 Detailed Description . . . 887
9.36.2 Member Data Documentation . . . 888
9.36.2.1 value . . . 888
9.36.2.2 compression_settings . . . 888
9.37 DDS_DataTags Struct Reference . . . 889
9.37.1 Detailed Description . . . 889
9.37.2 Member Data Documentation . . . 889
9.37.2.1 tags . . . 889
9.38 DDS_DataWriterCacheStatus Struct Reference . . . 889
9.38.1 Detailed Description . . . 890
9.38.2 Member Data Documentation . . . 890
9.38.2.1 sample_count_peak . . . 890
9.38.2.2 sample_count . . . 891
9.38.2.3 alive_instance_count . . . 891
9.38.2.4 alive_instance_count_peak . . . 891
9.38.2.5 disposed_instance_count . . . 891
9.38.2.6 disposed_instance_count_peak . . . 891

9.38.2.7 unregistered_instance_count . . . 892
9.38.2.8 unregistered_instance_count_peak . . . 892
9.39 DDS_DataWriterProtocolQosPolicy Struct Reference . . . 892
9.39.1 Detailed Description . . . 893
9.39.2 Member Data Documentation . . . 893
9.39.2.1 virtual_guid . . . 893
9.39.2.2 rtps_object_id . . . 894
9.39.2.3 push_on_write . . . 894
9.39.2.4 disable_positive_acks . . . 895
9.39.2.5 disable_inline_keyhash . . . 895
9.39.2.6 serialize_key_with_dispose . . . 896
9.39.2.7 propagate_app_ack_with_no_response . . . 896
9.39.2.8 rtps_reliable_writer . . . 897
9.39.2.9 initial_virtual_sequence_number . . . 897
9.40 DDS_DataWriterProtocolStatus Struct Reference . . . 897
9.40.1 Detailed Description . . . 900
9.40.2 Member Data Documentation . . . 900
9.40.2.1 pushed_sample_count . . . 900
9.40.2.2 pushed_sample_count_change . . . 900
9.40.2.3 pushed_sample_bytes . . . 900
9.40.2.4 pushed_sample_bytes_change . . . 901
9.40.2.5 filtered_sample_count . . . 901
9.40.2.6 filtered_sample_count_change . . . 901
9.40.2.7 filtered_sample_bytes . . . 901
9.40.2.8 filtered_sample_bytes_change . . . 901
9.40.2.9 sent_heartbeat_count . . . 902
9.40.2.10 sent_heartbeat_count_change . . . 902
9.40.2.11 sent_heartbeat_bytes . . . 902
9.40.2.12 sent_heartbeat_bytes_change . . . 902
9.40.2.13 pulled_sample_count . . . 902
9.40.2.14 pulled_sample_count_change . . . 903
9.40.2.15 pulled_sample_bytes . . . 903
9.40.2.16 pulled_sample_bytes_change . . . 903
9.40.2.17 received_ack_count . . . 903
9.40.2.18 received_ack_count_change . . . 903
9.40.2.19 received_ack_bytes . . . 904
9.40.2.20 received_ack_bytes_change . . . 904
9.40.2.21 received_nack_count . . . 904
9.40.2.22 received_nack_count_change . . . 904

9.40.2.23 received_nack_bytes . . . 904
9.40.2.24 received_nack_bytes_change . . . 904
9.40.2.25 sent_gap_count . . . 905
9.40.2.26 sent_gap_count_change . . . 905
9.40.2.27 sent_gap_bytes . . . 905
9.40.2.28 sent_gap_bytes_change . . . 905
9.40.2.29 rejected_sample_count . . . 905
9.40.2.30 rejected_sample_count_change . . . 905
9.40.2.31 send_window_size . . . 906
9.40.2.32 first_available_sample_sequence_number . . . 906
9.40.2.33 last_available_sample_sequence_number . . . 906
9.40.2.34 first_unacknowledged_sample_sequence_number . . . 906
9.40.2.35 first_available_sample_virtual_sequence_number . . . 906
9.40.2.36 last_available_sample_virtual_sequence_number . . . 907
9.40.2.37 first_unacknowledged_sample_virtual_sequence_number . . . 907
9.40.2.38 first_unacknowledged_sample_subscription_handle . . . 907
9.40.2.39 first_unelapsed_keep_duration_sample_sequence_number . . . 907
9.40.2.40 pushed_fragment_count . . . 907
9.40.2.41 pushed_fragment_bytes . . . 908
9.40.2.42 pulled_fragment_count . . . 908
9.40.2.43 pulled_fragment_bytes . . . 908
9.40.2.44 received_nack_fragment_count . . . 908
9.40.2.45 received_nack_fragment_bytes . . . 908
9.41 DDS_DataWriterQos Struct Reference . . . 909
9.41.1 Detailed Description . . . 911
9.41.2 Member Function Documentation . . . 911
9.41.2.1 operator==() . . . 911
9.41.2.2 operator"!=() . . . 912
9.41.3 Member Data Documentation . . . 912
9.41.3.1 durability . . . 912
9.41.3.2 durability_service . . . 912
9.41.3.3 deadline . . . 912
9.41.3.4 latency_budget . . . 913
9.41.3.5 liveliness . . . 913
9.41.3.6 reliability . . . 913
9.41.3.7 destination_order . . . 913
9.41.3.8 history . . . 913
9.41.3.9 resource_limits . . . 913
9.41.3.10 transport_priority . . . 914

9.41.3.11 lifespan . . . 914
9.41.3.12 user_data . . . 914
9.41.3.13 ownership . . . 914
9.41.3.14 ownership_strength . . . 914
9.41.3.15 writer_data_lifecycle . . . 914
9.41.3.16 representation . . . 915
9.41.3.17 data_tags . . . 915
9.41.3.18 writer_resource_limits . . . 915
9.41.3.19 protocol . . . 915
9.41.3.20 transport_selection . . . 915
9.41.3.21 unicast . . . 915
9.41.3.22 publish_mode . . . 916
9.41.3.23 property . . . 916
9.41.3.24 service . . . 916
9.41.3.25 batch . . . 916
9.41.3.26 multi_channel . . . 916
9.41.3.27 availability . . . 916
9.41.3.28 publication_name . . . 917
9.41.3.29 topic_query_dispatch . . . 917
9.41.3.30 transfer_mode . . . 917
9.41.3.31 type_support . . . 917
9.42 DDS_DataWriterResourceLimitsQosPolicy Struct Reference . . . 917
9.42.1 Detailed Description . . . 919
9.42.2 Member Data Documentation . . . 919
9.42.2.1 initial_concurrent_blocking_threads . . . 919
9.42.2.2 max_concurrent_blocking_threads . . . 920
9.42.2.3 max_remote_reader_filters . . . 920
9.42.2.4 initial_batches . . . 921
9.42.2.5 max_batches . . . 921
9.42.2.6 instance_replacement . . . 922
9.42.2.7 replace_empty_instances . . . 922
9.42.2.8 autoregister_instances . . . 923
9.42.2.9 initial_virtual_writers . . . 923
9.42.2.10 max_virtual_writers . . . 923
9.42.2.11 max_remote_readers . . . 924
9.42.2.12 max_app_ack_remote_readers . . . 924
9.42.2.13 initial_active_topic_queries . . . 924
9.42.2.14 max_active_topic_queries . . . 925
9.42.2.15 writer_loaned_sample_allocation . . . 925

9.42.2.16 initialize_writer_loaned_sample . . . 926
9.42.2.17 initial_remote_readers . . . 926
9.43 DDS_DataWriterShmemRefTransferModeSettings Struct Reference . . . 926
9.43.1 Detailed Description . . . 927
9.43.2 Member Data Documentation . . . 927
9.43.2.1 enable_data_consistency_check . . . 927
9.44 DDS_DataWriterTransferModeQosPolicy Struct Reference . . . 927
9.44.1 Detailed Description . . . 928
9.44.2 Member Data Documentation . . . 928
9.44.2.1 shmem_ref_settings . . . 928
9.45 DDS_DeadlineQosPolicy Struct Reference . . . 928
9.45.1 Detailed Description . . . 929
9.45.2 Usage . . . 929
9.45.3 Compatibility . . . 930
9.45.4 Consistency . . . 930
9.45.5 Member Data Documentation . . . 930
9.45.5.1 period . . . 930
9.46 DDS_DestinationOrderQosPolicy Struct Reference . . . 931
9.46.1 Detailed Description . . . 931
9.46.2 Usage . . . 931
9.46.3 Compatibility . . . 932
9.46.4 Member Data Documentation . . . 933
9.46.4.1 kind . . . 933
9.46.4.2 scope . . . 933
9.46.4.3 source_timestamp_tolerance . . . 933
9.47 DDS_DiscoveryConfigQosPolicy Struct Reference . . . 934
9.47.1 Detailed Description . . . 936
9.47.2 Member Data Documentation . . . 937
9.47.2.1 participant_liveliness_lease_duration . . . 937
9.47.2.2 participant_liveliness_assert_period . . . 937
9.47.2.3 participant_announcement_period . . . 938
9.47.2.4 remote_participant_purge_kind . . . 938
9.47.2.5 max_liveliness_loss_detection_period . . . 939
9.47.2.6 initial_participant_announcements . . . 939
9.47.2.7 new_remote_participant_announcements . . . 940
9.47.2.8 min_initial_participant_announcement_period . . . 940
9.47.2.9 max_initial_participant_announcement_period . . . 941
9.47.2.10 participant_reader_resource_limits . . . 941
9.47.2.11 publication_reader . . . 942

9.47.2.12 publication_reader_resource_limits . . . 942
9.47.2.13 subscription_reader . . . 943
9.47.2.14 subscription_reader_resource_limits . . . 943
9.47.2.15 publication_writer . . . 944
9.47.2.16 publication_writer_data_lifecycle . . . 944
9.47.2.17 subscription_writer . . . 945
9.47.2.18 subscription_writer_data_lifecycle . . . 946
9.47.2.19 builtin_discovery_plugins . . . 946
9.47.2.20 enabled_builtin_channels . . . 946
9.47.2.21 participant_message_reader_reliability_kind . . . 947
9.47.2.22 participant_message_reader . . . 947
9.47.2.23 participant_message_writer . . . 948
9.47.2.24 publication_writer_publish_mode . . . 948
9.47.2.25 subscription_writer_publish_mode . . . 949
9.47.2.26 asynchronous_publisher . . . 949
9.47.2.27 default_domain_announcement_period . . . 950
9.47.2.28 ignore_default_domain_announcements . . . 950
9.47.2.29 service_request_writer . . . 951
9.47.2.30 service_request_writer_data_lifecycle . . . 951
9.47.2.31 service_request_writer_publish_mode . . . 952
9.47.2.32 service_request_reader . . . 952
9.47.2.33 locator_reachability_assert_period . . . 953
9.47.2.34 locator_reachability_lease_duration . . . 953
9.47.2.35 locator_reachability_change_detection_period . . . 954
9.47.2.36 secure_volatile_writer . . . 954
9.47.2.37 secure_volatile_writer_publish_mode . . . 955
9.47.2.38 secure_volatile_reader . . . 955
9.47.2.39 endpoint_type_object_lb_serialization_threshold . . . 956
9.47.2.40 dns_tracker_polling_period . . . 956
9.47.2.41 participant_configuration_writer_publish_mode . . . 957
9.47.2.42 participant_configuration_writer . . . 958
9.47.2.43 participant_configuration_writer_data_lifecycle . . . 958
9.47.2.44 participant_configuration_reader . . . 959
9.47.2.45 participant_configuration_reader_resource_limits . . . 959
9.47.2.46 request_types_filter . . . 960
9.48 DDS_DiscoveryQosPolicy Struct Reference . . . 960
9.48.1 Detailed Description . . . 961
9.48.2 Usage . . . 961
9.48.3 Member Data Documentation . . . 961

9.48.3.1 enabled_transports . . . 961
9.48.3.2 initial_peers . . . 962
9.48.3.3 multicast_receive_addresses . . . 963
9.48.3.4 metatraffic_transport_priority . . . 963
9.48.3.5 accept_unknown_peers . . . 964
9.48.3.6 enable_endpoint_discovery . . . 964
9.49 DDS_DomainParticipantConfigParams_t Struct Reference . . . 964
9.49.1 Detailed Description . . . 965
9.49.2 Member Data Documentation . . . 965
9.49.2.1 domain_id . . . 965
9.49.2.2 participant_name . . . 965
9.49.2.3 participant_qos_library_name . . . 966
9.49.2.4 participant_qos_profile_name . . . 966
9.49.2.5 domain_entity_qos_library_name . . . 966
9.49.2.6 domain_entity_qos_profile_name . . . 966
9.50 DDS_DomainParticipantFactoryQos Struct Reference . . . 967
9.50.1 Detailed Description . . . 968
9.50.2 Member Function Documentation . . . 968
9.50.2.1 operator==() . . . 968
9.50.2.2 operator"!=() . . . 968
9.50.3 Member Data Documentation . . . 969
9.50.3.1 entity_factory . . . 969
9.50.3.2 resource_limits . . . 969
9.50.3.3 profile . . . 969
9.50.3.4 logging . . . 969
9.50.3.5 monitoring . . . 969
9.51 DDS_DomainParticipantProtocolStatus Struct Reference . . . 970
9.51.1 Detailed Description . . . 970
9.51.2 Member Data Documentation . . . 970
9.51.2.1 corrupted_rtps_message_count . . . 970
9.51.2.2 corrupted_rtps_message_count_change . . . 970
9.51.2.3 last_corrupted_message_timestamp . . . 971
9.52 DDS_DomainParticipantQos Struct Reference . . . 971
9.52.1 Detailed Description . . . 972
9.52.2 Member Function Documentation . . . 973
9.52.2.1 operator==() . . . 973
9.52.2.2 operator"!=() . . . 973
9.52.3 Member Data Documentation . . . 973
9.52.3.1 user_data . . . 974

9.52.3.2 entity_factory . . . 974
9.52.3.3 wire_protocol . . . 974
9.52.3.4 transport_builtin . . . 974
9.52.3.5 default_unicast . . . 974
9.52.3.6 discovery . . . 974
9.52.3.7 resource_limits . . . 975
9.52.3.8 event . . . 975
9.52.3.9 receiver_pool . . . 975
9.52.3.10 database . . . 975
9.52.3.11 discovery_config . . . 975
9.52.3.12 property . . . 976
9.52.3.13 participant_name . . . 976
9.52.3.14 multicast_mapping . . . 976
9.52.3.15 service . . . 976
9.52.3.16 partition . . . 976
9.52.3.17 type_support . . . 976
9.53 DDS_DomainParticipantResourceLimitsQosPolicy Struct Reference . . . 977
9.53.1 Detailed Description . . . 980
9.53.2 Member Data Documentation . . . 981
9.53.2.1 local_writer_allocation . . . 981
9.53.2.2 local_reader_allocation . . . 981
9.53.2.3 local_publisher_allocation . . . 981
9.53.2.4 local_subscriber_allocation . . . 982
9.53.2.5 local_topic_allocation . . . 982
9.53.2.6 remote_writer_allocation . . . 982
9.53.2.7 remote_reader_allocation . . . 983
9.53.2.8 remote_participant_allocation . . . 983
9.53.2.9 matching_writer_reader_pair_allocation . . . 983
9.53.2.10 matching_reader_writer_pair_allocation . . . 984
9.53.2.11 ignored_entity_allocation . . . 984
9.53.2.12 content_filtered_topic_allocation . . . 984
9.53.2.13 content_filter_allocation . . . 985
9.53.2.14 read_condition_allocation . . . 985
9.53.2.15 query_condition_allocation . . . 985
9.53.2.16 outstanding_asynchronous_sample_allocation . . . 986
9.53.2.17 flow_controller_allocation . . . 986
9.53.2.18 local_writer_hash_buckets . . . 986
9.53.2.19 local_reader_hash_buckets . . . 987
9.53.2.20 local_publisher_hash_buckets . . . 987

9.53.2.21 local_subscriber_hash_buckets . . . 987
9.53.2.22 local_topic_hash_buckets . . . 988
9.53.2.23 remote_writer_hash_buckets . . . 988
9.53.2.24 remote_reader_hash_buckets . . . 988
9.53.2.25 remote_participant_hash_buckets . . . 989
9.53.2.26 matching_writer_reader_pair_hash_buckets . . . 989
9.53.2.27 matching_reader_writer_pair_hash_buckets . . . 989
9.53.2.28 ignored_entity_hash_buckets . . . 990
9.53.2.29 content_filtered_topic_hash_buckets . . . 990
9.53.2.30 content_filter_hash_buckets . . . 990
9.53.2.31 flow_controller_hash_buckets . . . 991
9.53.2.32 max_gather_destinations . . . 991
9.53.2.33 participant_user_data_max_length . . . 991
9.53.2.34 topic_data_max_length . . . 992
9.53.2.35 publisher_group_data_max_length . . . 992
9.53.2.36 subscriber_group_data_max_length . . . 992
9.53.2.37 writer_user_data_max_length . . . 993
9.53.2.38 reader_user_data_max_length . . . 993
9.53.2.39 max_partitions . . . 993
9.53.2.40 max_partition_cumulative_characters . . . 994
9.53.2.41 type_code_max_serialized_length . . . 994
9.53.2.42 type_object_max_serialized_length . . . 995
9.53.2.43 serialized_type_object_dynamic_allocation_threshold . . . 995
9.53.2.44 type_object_max_deserialized_length . . . 996
9.53.2.45 deserialized_type_object_dynamic_allocation_threshold . . . 997
9.53.2.46 contentfilter_property_max_length . . . 997
9.53.2.47 channel_seq_max_length . . . 998
9.53.2.48 channel_filter_expression_max_length . . . 998
9.53.2.49 participant_property_list_max_length . . . 998
9.53.2.50 participant_property_string_max_length . . . 999
9.53.2.51 writer_property_list_max_length . . . 999
9.53.2.52 writer_property_string_max_length . . . 999
9.53.2.53 reader_property_list_max_length . . . 1000
9.53.2.54 reader_property_string_max_length . . . 1000
9.53.2.55 max_endpoint_groups . . . 1000
9.53.2.56 max_endpoint_group_cumulative_characters . . . 1001
9.53.2.57 transport_info_list_max_length . . . 1001
9.53.2.58 ignored_entity_replacement_kind . . . 1001
9.53.2.59 remote_topic_query_allocation . . . 1002

9.53.2.60 remote_topic_query_hash_buckets . . . 1002
9.53.2.61 writer_data_tag_list_max_length . . . 1003
9.53.2.62 writer_data_tag_string_max_length . . . 1003
9.53.2.63 reader_data_tag_list_max_length . . . 1003
9.53.2.64 reader_data_tag_string_max_length . . . 1004
9.53.2.65 shmem_ref_transfer_mode_max_segments . . . 1004
9.54 DDS_DoubleSeq Struct Reference . . . 1004
9.54.1 Detailed Description . . . 1004
9.55 DDS_DurabilityQosPolicy Struct Reference . . . 1005
9.55.1 Detailed Description . . . 1005
9.55.2 Usage . . . 1006
9.55.2.1 Transient and Persistent Durability . . . 1006
9.55.3 Compatibility . . . 1007
9.55.4 Member Data Documentation . . . 1007
9.55.4.1 kind . . . 1007
9.55.4.2 direct_communication . . . 1008
9.55.4.3 writer_depth . . . 1008
9.55.4.4 storage_settings . . . 1009
9.56 DDS_DurabilityServiceQosPolicy Struct Reference . . . 1009
9.56.1 Detailed Description . . . 1010
9.56.2 Usage . . . 1010
9.56.3 Member Data Documentation . . . 1010
9.56.3.1 service_cleanup_delay . . . 1010
9.56.3.2 history_kind . . . 1011
9.56.3.3 history_depth . . . 1011
9.56.3.4 max_samples . . . 1011
9.56.3.5 max_instances . . . 1012
9.56.3.6 max_samples_per_instance . . . 1012
9.57 DDS_Duration_t Struct Reference . . . 1012
9.57.1 Detailed Description . . . 1013
9.57.2 Member Data Documentation . . . 1013
9.57.2.1 sec . . . 1013
9.57.2.2 nanosec . . . 1013
9.58 DDS_DynamicData Struct Reference . . . 1013
9.58.1 Detailed Description . . . 1022
9.58.2 Member Names and IDs . . . 1022
9.58.2.1 Getting and Setting Union Discriminator Values . . . 1022
9.58.2.2 Hierarchical Member Names . . . 1023
9.58.3 Arrays and Sequences . . . 1024

9.58.4 Available Functionality . . . 1024
9.58.4.1 Lifecycle and Utility Methods . . . 1024
9.58.4.2 Getters and Setters . . . 1025
9.58.4.3 Query and Iteration . . . 1027
9.58.4.4 Type/Object Association . . . 1028
9.58.4.5 Performance Considerations . . . 1028
9.58.5 Constructor & Destructor Documentation . . . 1029
9.58.5.1 DDS_DynamicData() . . . 1029
9.58.5.2 ~DDS_DynamicData() . . . 1030
9.58.6 Member Function Documentation . . . 1030
9.58.6.1 is_valid() . . . 1031
9.58.6.2 copy() . . . 1031
9.58.6.3 equal() . . . 1032
9.58.6.4 operator=() . . . 1032
9.58.6.5 operator==() . . . 1033
9.58.6.6 clear_all_members() . . . 1033
9.58.6.7 clear_optional_member() . . . 1033
9.58.6.8 clear_member() . . . 1034
9.58.6.9 is_cdr() . . . 1035
9.58.6.10 get_cdr_buffer() . . . 1036
9.58.6.11 set_cdr_buffer() . . . 1036
9.58.6.12 from_cdr_buffer() . . . 1037
9.58.6.13 to_cdr_buffer() . . . 1037
9.58.6.14 to_cdr_buffer_ex() . . . 1038
9.58.6.15 to_string() . . . 1038
9.58.6.16 from_string() . . . 1039
9.58.6.17 print() . . . 1040
9.58.6.18 get_info() . . . 1041
9.58.6.19 bind_type() . . . 1041
9.58.6.20 unbind_type() . . . 1042
9.58.6.21 bind_complex_member() . . . 1042
9.58.6.22 unbind_complex_member() . . . 1044
9.58.6.23 get_type() . . . 1045
9.58.6.24 get_type_kind() . . . 1045
9.58.6.25 get_member_count() . . . 1046
9.58.6.26 member_exists() . . . 1046
9.58.6.27 member_exists_in_type() . . . 1047
9.58.6.28 get_member_info() . . . 1048
9.58.6.29 get_member_info_by_index() . . . 1049

9.58.6.30 get_member_type() . . . 1050
9.58.6.31 is_member_key() . . . 1051
9.58.6.32 get_long() . . . 1052
9.58.6.33 get_short() . . . 1052
9.58.6.34 get_ulong() . . . 1053
9.58.6.35 get_ushort() . . . 1054
9.58.6.36 get_float() . . . 1055
9.58.6.37 get_double() . . . 1056
9.58.6.38 get_boolean() . . . 1056
9.58.6.39 get_char() . . . 1057
9.58.6.40 get_octet() . . . 1058
9.58.6.41 get_longlong() . . . 1059
9.58.6.42 get_ulonglong() . . . 1060
9.58.6.43 get_longdouble() . . . 1061
9.58.6.44 get_wchar() . . . 1061
9.58.6.45 get_string() . . . 1062
9.58.6.46 get_wstring() . . . 1063
9.58.6.47 get_int8() . . . 1064
9.58.6.48 get_uint8() . . . 1065
9.58.6.49 get_complex_member() . . . 1066
9.58.6.50 get_discriminator() . . . 1067
9.58.6.51 get_long_array() . . . 1068
9.58.6.52 get_short_array() . . . 1068
9.58.6.53 get_ulong_array() . . . 1069
9.58.6.54 get_ushort_array() . . . 1070
9.58.6.55 get_float_array() . . . 1071
9.58.6.56 get_double_array() . . . 1072
9.58.6.57 get_boolean_array() . . . 1073
9.58.6.58 get_char_array() . . . 1074
9.58.6.59 get_octet_array() . . . 1075
9.58.6.60 get_longlong_array() . . . 1076
9.58.6.61 get_ulonglong_array() . . . 1076
9.58.6.62 get_longdouble_array() . . . 1077
9.58.6.63 get_wchar_array() . . . 1078
9.58.6.64 get_int8_array() . . . 1079
9.58.6.65 get_uint8_array() . . . 1080
9.58.6.66 get_long_seq() . . . 1081
9.58.6.67 get_short_seq() . . . 1082
9.58.6.68 get_ulong_seq() . . . 1083

9.58.6.69 get_ushort_seq() . . . 1083
9.58.6.70 get_float_seq() . . . 1084
9.58.6.71 get_double_seq() . . . 1085
9.58.6.72 get_boolean_seq() . . . 1086
9.58.6.73 get_char_seq() . . . 1087
9.58.6.74 get_octet_seq() . . . 1088
9.58.6.75 get_longlong_seq() . . . 1088
9.58.6.76 get_ulonglong_seq() . . . 1089
9.58.6.77 get_longdouble_seq() . . . 1090
9.58.6.78 get_wchar_seq() . . . 1091
9.58.6.79 get_int8_seq() . . . 1092
9.58.6.80 get_uint8_seq() . . . 1093
9.58.6.81 set_long() . . . 1093
9.58.6.82 set_short() . . . 1094
9.58.6.83 set_ulong() . . . 1095
9.58.6.84 set_ushort() . . . 1096
9.58.6.85 set_float() . . . 1096
9.58.6.86 set_double() . . . 1097
9.58.6.87 set_boolean() . . . 1098
9.58.6.88 set_char() . . . 1098
9.58.6.89 set_octet() . . . 1099
9.58.6.90 set_longlong() . . . 1100
9.58.6.91 set_ulonglong() . . . 1101
9.58.6.92 set_longdouble() . . . 1101
9.58.6.93 set_wchar() . . . 1102
9.58.6.94 set_string() . . . 1103
9.58.6.95 set_wstring() . . . 1103
9.58.6.96 set_int8() . . . 1104
9.58.6.97 set_uint8() . . . 1105
9.58.6.98 set_complex_member() . . . 1106
9.58.6.99 set_long_array() . . . 1107
9.58.6.100 set_short_array() . . . 1108
9.58.6.101 set_ulong_array() . . . 1109
9.58.6.102 set_ushort_array() . . . 1110
9.58.6.103 set_float_array() . . . 1110
9.58.6.104 set_double_array() . . . 1111
9.58.6.105 set_boolean_array() . . . 1112
9.58.6.106 set_char_array() . . . 1113
9.58.6.107 set_octet_array() . . . 1114

9.58.6.108 set_longlong_array() . . . 1115
9.58.6.109 set_ulonglong_array() . . . 1116
9.58.6.110 set_longdouble_array() . . . 1116
9.58.6.111 set_wchar_array() . . . 1117
9.58.6.112 set_int8_array() . . . 1118
9.58.6.113 set_uint8_array() . . . 1119
9.58.6.114 set_long_seq() . . . 1120
9.58.6.115 set_short_seq() . . . 1121
9.58.6.116 set_ulong_seq() . . . 1121
9.58.6.117 set_ushort_seq() . . . 1122
9.58.6.118 set_float_seq() . . . 1123
9.58.6.119 set_double_seq() . . . 1124
9.58.6.120 set_boolean_seq() . . . 1125
9.58.6.121 set_char_seq() . . . 1125
9.58.6.122 set_octet_seq() . . . 1126
9.58.6.123 set_longlong_seq() . . . 1127
9.58.6.124 set_ulonglong_seq() . . . 1128
9.58.6.125 set_longdouble_seq() . . . 1129
9.58.6.126 set_wchar_seq() . . . 1129
9.58.6.127 set_int8_seq() . . . 1130
9.58.6.128 set_uint8_seq() . . . 1131
9.58.6.129 set_discriminator() . . . 1132
9.59 DDS_DynamicDataInfo Struct Reference . . . 1133
9.59.1 Detailed Description . . . 1134
9.59.2 Member Data Documentation . . . 1134
9.59.2.1 member_count . . . 1134
9.59.2.2 stored_size . . . 1134
9.60 DDS_DynamicDataJsonParserProperties_t Struct Reference . . . 1134
9.60.1 Detailed Description . . . 1134
9.61 DDS_DynamicDataMemberInfo Struct Reference . . . 1135
9.61.1 Detailed Description . . . 1135
9.61.2 Member Data Documentation . . . 1135
9.61.2.1 member_id . . . 1135
9.61.2.2 member_name . . . 1136
9.61.2.3 member_exists . . . 1136
9.61.2.4 member_kind . . . 1136
9.61.2.5 element_count . . . 1136
9.61.2.6 element_kind . . . 1137
9.62 DDS_DynamicDataProperty_t Struct Reference . . . 1137

9.62.1 Detailed Description . . . 1137
9.62.2 Member Data Documentation . . . 1137
9.62.2.1 buffer_initial_size . . . 1137
9.62.2.2 buffer_max_size . . . 1138
9.63 DDS_DynamicDataSeq Struct Reference . . . 1138
9.63.1 Detailed Description . . . 1138
9.64 DDS_DynamicDataTypeProperty_t Struct Reference . . . 1138
9.64.1 Detailed Description . . . 1139
9.64.2 Member Data Documentation . . . 1139
9.64.2.1 data . . . 1139
9.64.2.2 serialization . . . 1139
9.65 DDS_DynamicDataTypeSerializationProperty_t Struct Reference . . . 1139
9.65.1 Detailed Description . . . 1140
9.65.2 Member Data Documentation . . . 1140
9.65.2.1 use_42e_compatible_alignment . . . 1140
9.65.2.2 max_size_serialized . . . 1140
9.65.2.3 min_size_serialized . . . 1141
9.65.2.4 trim_to_size . . . 1141
9.65.2.5 skip_deserialization . . . 1141
9.66 DDS_EndpointGroup_t Struct Reference . . . 1142
9.66.1 Detailed Description . . . 1142
9.66.2 Member Data Documentation . . . 1142
9.66.2.1 role_name . . . 1142
9.66.2.2 quorum_count . . . 1142
9.67 DDS_EndpointGroupSeq Struct Reference . . . 1142
9.67.1 Detailed Description . . . 1143
9.68 DDS_EndpointTrustAlgorithmInfo Struct Reference . . . 1143
9.68.1 Detailed Description . . . 1143
9.68.2 Member Data Documentation . . . 1143
9.68.2.1 interceptor . . . 1143
9.69 DDS_EndpointTrustInterceptorAlgorithmInfo Struct Reference . . . 1144
9.69.1 Detailed Description . . . 1144
9.69.2 Member Data Documentation . . . 1144
9.69.2.1 required_mask . . . 1144
9.69.2.2 supported_mask . . . 1144
9.70 DDS_EndpointTrustProtectionInfo Struct Reference . . . 1144
9.70.1 Detailed Description . . . 1145
9.70.2 Member Data Documentation . . . 1145
9.70.2.1 bitmask . . . 1145

9.70.2.2 plugin_bitmask . . . 1145
9.71 DDS_EntityFactoryQosPolicy Struct Reference . . . 1145
9.71.1 Detailed Description . . . 1146
9.71.2 Usage . . . 1146
9.71.3 Member Data Documentation . . . 1147
9.71.3.1 autoenable_created_entities . . . 1147
9.72 DDS_EntityNameQosPolicy Struct Reference . . . 1147
9.72.1 Detailed Description . . . 1148
9.72.2 Usage . . . 1148
9.72.3 Member Data Documentation . . . 1148
9.72.3.1 name . . . 1148
9.72.3.2 role_name . . . 1149
9.73 DDS_EnumMember Struct Reference . . . 1149
9.73.1 Detailed Description . . . 1149
9.73.2 Member Data Documentation . . . 1149
9.73.2.1 name . . . 1150
9.73.2.2 ordinal . . . 1150
9.74 DDS_EnumMemberSeq Struct Reference . . . 1150
9.74.1 Detailed Description . . . 1150
9.75 DDS_EventQosPolicy Struct Reference . . . 1150
9.75.1 Detailed Description . . . 1151
9.75.2 Member Data Documentation . . . 1151
9.75.2.1 thread . . . 1151
9.75.2.2 initial_count . . . 1152
9.75.2.3 max_count . . . 1152
9.76 DDS_ExpressionProperty Struct Reference . . . 1152
9.76.1 Detailed Description . . . 1153
9.76.2 Member Data Documentation . . . 1153
9.76.2.1 key_only_filter . . . 1153
9.76.2.2 writer_side_filter_optimization . . . 1153
9.77 DDS_FilterSampleInfo Struct Reference . . . 1153
9.77.1 Detailed Description . . . 1154
9.77.2 Member Data Documentation . . . 1154
9.77.2.1 related_sample_identity . . . 1154
9.77.2.2 related_source_guid . . . 1154
9.77.2.3 related_reader_guid . . . 1155
9.77.2.4 sample_identity . . . 1155
9.77.2.5 source_guid . . . 1155
9.78 DDS_FloatSeq Struct Reference . . . 1156

9.78.1 Detailed Description . . . 1156
9.79 DDS_FlowControllerProperty_t Struct Reference . . . 1156
9.79.1 Detailed Description . . . 1156
9.79.2 Member Data Documentation . . . 1157
9.79.2.1 scheduling_policy . . . 1157
9.79.2.2 token_bucket . . . 1157
9.80 DDS_FlowControllerTokenBucketProperty_t Struct Reference . . . 1157
9.80.1 Detailed Description . . . 1158
9.80.2 Member Data Documentation . . . 1158
9.80.2.1 max_tokens . . . 1158
9.80.2.2 tokens_added_per_period . . . 1159
9.80.2.3 tokens_leaked_per_period . . . 1159
9.80.2.4 period . . . 1159
9.80.2.5 bytes_per_token . . . 1160
9.81 DDS_GroupDataQosPolicy Struct Reference . . . 1160
9.81.1 Detailed Description . . . 1160
9.81.2 Usage . . . 1161
9.81.3 Member Data Documentation . . . 1161
9.81.3.1 value . . . 1161
9.82 DDS_GUID_t Struct Reference . . . 1161
9.82.1 Detailed Description . . . 1162
9.82.2 Member Data Documentation . . . 1162
9.82.2.1 value . . . 1162
9.83 DDS_HistoryQosPolicy Struct Reference . . . 1162
9.83.1 Detailed Description . . . 1163
9.83.2 Usage . . . 1163
9.83.3 Consistency . . . 1164
9.83.4 Member Data Documentation . . . 1164
9.83.4.1 kind . . . 1164
9.83.4.2 depth . . . 1165
9.84 DDS_InconsistentTopicStatus Struct Reference . . . 1165
9.84.1 Detailed Description . . . 1166
9.84.2 Member Data Documentation . . . 1166
9.84.2.1 total_count . . . 1166
9.84.2.2 total_count_change . . . 1166
9.85 DDS_InstanceHandleSeq Struct Reference . . . 1166
9.85.1 Detailed Description . . . 1167
9.86 DDS_Int8Seq Struct Reference . . . 1167
9.86.1 Detailed Description . . . 1167

9.87 DDS_InvalidLocalAdvanceNoticeStatus Struct Reference . . . 1167
9.87.1 Detailed Description . . . 1168
9.87.2 Member Data Documentation . . . 1168
9.87.2.1 expiration_time . . . 1168
9.87.2.2 expiration_code . . . 1168
9.87.2.3 expiration_message . . . 1168
9.88 DDS_KeyedOctets Struct Reference . . . 1168
9.88.1 Detailed Description . . . 1169
9.88.2 Constructor & Destructor Documentation . . . 1169
9.88.2.1 DDS_KeyedOctets() [1/2] . . . 1169
9.88.2.2 DDS_KeyedOctets() [2/2] . . . 1169
9.88.2.3 ∼DDS_KeyedOctets() . . . 1170
9.88.3 Member Data Documentation . . . 1170
9.88.3.1 key . . . 1170
9.88.3.2 length . . . 1170
9.88.3.3 value . . . 1170
9.89 DDS_KeyedOctetsSeq Struct Reference . . . 1170
9.89.1 Detailed Description . . . 1171
9.90 DDS_KeyedString Struct Reference . . . 1171
9.90.1 Detailed Description . . . 1171
9.90.2 Constructor & Destructor Documentation . . . 1171
9.90.2.1 DDS_KeyedString() [1/2] . . . 1172
9.90.2.2 DDS_KeyedString() [2/2] . . . 1172
9.90.2.3 ∼DDS_KeyedString() . . . 1172
9.90.3 Member Data Documentation . . . 1172
9.90.3.1 key . . . 1173
9.90.3.2 value . . . 1173
9.91 DDS_KeyedStringSeq Struct Reference . . . 1173
9.91.1 Detailed Description . . . 1173
9.92 DDS_LatencyBudgetQosPolicy Struct Reference . . . 1173
9.92.1 Detailed Description . . . 1174
9.92.2 Usage . . . 1174
9.92.3 Compatibility . . . 1174
9.92.4 Member Data Documentation . . . 1175
9.92.4.1 duration . . . 1175
9.93 DDS_LifespanQosPolicy Struct Reference . . . 1175
9.93.1 Detailed Description . . . 1175
9.93.2 Usage . . . 1176
9.93.3 Member Data Documentation . . . 1176

9.93.3.1 duration . . . 1176
9.94 DDS_LivelinessChangedStatus Struct Reference . . . 1176
9.94.1 Detailed Description . . . 1177
9.94.2 Member Data Documentation . . . 1178
9.94.2.1 alive_count . . . 1178
9.94.2.2 not_alive_count . . . 1178
9.94.2.3 alive_count_change . . . 1178
9.94.2.4 not_alive_count_change . . . 1178
9.94.2.5 last_publication_handle . . . 1179
9.95 DDS_LivelinessLostStatus Struct Reference . . . 1179
9.95.1 Detailed Description . . . 1179
9.95.2 Member Data Documentation . . . 1179
9.95.2.1 total_count . . . 1180
9.95.2.2 total_count_change . . . 1180
9.96 DDS_LivelinessQosPolicy Struct Reference . . . 1180
9.96.1 Detailed Description . . . 1180
9.96.2 Usage . . . 1181
9.96.3 Compatibility . . . 1182
9.96.4 Member Data Documentation . . . 1182
9.96.4.1 kind . . . 1182
9.96.4.2 lease_duration . . . 1183
9.96.4.3 assertions_per_lease_duration . . . 1183
9.97 DDS_Locator_t Struct Reference . . . 1184
9.97.1 Detailed Description . . . 1184
9.97.2 Member Data Documentation . . . 1184
9.97.2.1 kind . . . 1184
9.97.2.2 port . . . 1184
9.97.2.3 address . . . 1185
9.98 DDS_LocatorFilter_t Struct Reference . . . 1185
9.98.1 Detailed Description . . . 1185
9.98.2 Member Data Documentation . . . 1185
9.98.2.1 locators . . . 1185
9.98.2.2 filter_expression . . . 1186
9.99 DDS_LocatorFilterQosPolicy Struct Reference . . . 1186
9.99.1 Detailed Description . . . 1186
9.99.2 Member Data Documentation . . . 1187
9.99.2.1 locator_filters . . . 1187
9.99.2.2 filter_name . . . 1187
9.100 DDS_LocatorFilterSeq Struct Reference . . . 1187

9.100.1 Detailed Description . . . 1187
9.101 DDS_LocatorSeq Struct Reference . . . 1188
9.101.1 Detailed Description . . . 1188
9.102 DDS_LoggingQosPolicy Struct Reference . . . 1188
9.102.1 Detailed Description . . . 1188
9.102.2 Member Data Documentation . . . 1189
9.102.2.1 verbosity . . . 1189
9.102.2.2 category . . . 1189
9.102.2.3 print_format . . . 1189
9.102.2.4 output_file . . . 1190
9.102.2.5 output_file_suffix . . . 1190
9.102.2.6 max_bytes_per_file . . . 1191
9.102.2.7 max_files . . . 1191
9.103 DDS_LongDoubleSeq Struct Reference . . . 1192
9.103.1 Detailed Description . . . 1192
9.104 DDS_LongLongSeq Struct Reference . . . 1192
9.104.1 Detailed Description . . . 1192
9.105 DDS_LongSeq Struct Reference . . . 1192
9.105.1 Detailed Description . . . 1193
9.106 DDS_MonitoringDedicatedParticipantSettings Struct Reference . . . 1193
9.106.1 Detailed Description . . . 1193
9.106.2 Member Data Documentation . . . 1193
9.106.2.1 enable . . . 1194
9.106.2.2 domain_id . . . 1194
9.106.2.3 participant_qos_profile_name . . . 1194
9.106.2.4 collector_initial_peers . . . 1195
9.107 DDS_MonitoringDistributionSettings Struct Reference . . . 1195
9.107.1 Detailed Description . . . 1196
9.107.2 Member Data Documentation . . . 1196
9.107.2.1 dedicated_participant . . . 1196
9.107.2.2 publisher_qos_profile_name . . . 1196
9.107.2.3 event_settings . . . 1197
9.107.2.4 periodic_settings . . . 1197
9.107.2.5 logging_settings . . . 1197
9.108 DDS_MonitoringEventDistributionSettings Struct Reference . . . 1197
9.108.1 Detailed Description . . . 1198
9.108.2 Member Data Documentation . . . 1198
9.108.2.1 concurrency_level . . . 1199
9.108.2.2 datawriter_qos_profile_name . . . 1199

9.108.2.3 thread . . . 1200
9.108.2.4 publication_period . . . 1200
9.109 DDS_MonitoringLoggingDistributionSettings Struct Reference . . . 1200
9.109.1 Detailed Description . . . 1201
9.109.2 Member Data Documentation . . . 1201
9.109.2.1 concurrency_level . . . 1202
9.109.2.2 max_historical_logs . . . 1202
9.109.2.3 datawriter_qos_profile_name . . . 1203
9.109.2.4 thread . . . 1203
9.109.2.5 publication_period . . . 1203
9.110 DDS_MonitoringLoggingForwardingSettings Struct Reference . . . 1204
9.110.1 Detailed Description . . . 1204
9.110.2 Member Data Documentation . . . 1204
9.110.2.1 middleware_forwarding_level . . . 1204
9.110.2.2 security_event_forwarding_level . . . 1205
9.110.2.3 service_forwarding_level . . . 1205
9.110.2.4 user_forwarding_level . . . 1205
9.111 DDS_MonitoringMetricSelection Struct Reference . . . 1206
9.111.1 Detailed Description . . . 1206
9.111.2 Member Data Documentation . . . 1206
9.111.2.1 resource_selection . . . 1206
9.111.2.2 enabled_metrics_selection . . . 1207
9.111.2.3 disabled_metrics_selection . . . 1208
9.112 DDS_MonitoringMetricSelectionSeq Struct Reference . . . 1208
9.112.1 Detailed Description . . . 1208
9.113 DDS_MonitoringPeriodicDistributionSettings Struct Reference . . . 1209
9.113.1 Detailed Description . . . 1209
9.113.2 Member Data Documentation . . . 1209
9.113.2.1 datawriter_qos_profile_name . . . 1209
9.113.2.2 thread . . . 1210
9.113.2.3 polling_period . . . 1210
9.114 DDS_MonitoringQosPolicy Struct Reference . . . 1210
9.114.1 Detailed Description . . . 1211
9.114.2 Member Data Documentation . . . 1211
9.114.2.1 enable . . . 1211
9.114.2.2 application_name . . . 1212
9.114.2.3 distribution_settings . . . 1212
9.114.2.4 telemetry_data . . . 1212
9.115 DDS_MonitoringTelemetryData Struct Reference . . . 1213

9.115.1 Detailed Description . . . 1213
9.115.2 Member Data Documentation . . . 1213
9.115.2.1 metrics . . . 1213
9.115.2.2 logs . . . 1214
9.116 DDS_MultiChannelQosPolicy Struct Reference . . . 1214
9.116.1 Detailed Description . . . 1214
9.116.2 Usage . . . 1215
9.116.3 Member Data Documentation . . . 1215
9.116.3.1 channels . . . 1215
9.116.3.2 filter_name . . . 1216
9.117 DDS_Octets Struct Reference . . . 1217
9.117.1 Detailed Description . . . 1217
9.117.2 Constructor & Destructor Documentation . . . 1217
9.117.2.1 DDS_Octets() [1/2] . . . 1217
9.117.2.2 DDS_Octets() [2/2] . . . 1217
9.117.2.3 ~DDS_Octets() . . . 1218
9.117.3 Member Data Documentation . . . 1218
9.117.3.1 length . . . 1218
9.117.3.2 value . . . 1218
9.118 DDS_OctetSeq Struct Reference . . . 1218
9.118.1 Detailed Description . . . 1219
9.119 DDS_OctetsSeq Struct Reference . . . 1219
9.119.1 Detailed Description . . . 1219
9.120 DDS_OfferedDeadlineMissedStatus Struct Reference . . . 1219
9.120.1 Detailed Description . . . 1220
9.120.2 Member Data Documentation . . . 1220
9.120.2.1 total_count . . . 1220
9.120.2.2 total_count_change . . . 1220
9.120.2.3 last_instance_handle . . . 1220
9.121 DDS_OfferedIncompatibleQosStatus Struct Reference . . . 1221
9.121.1 Detailed Description . . . 1221
9.121.2 Member Data Documentation . . . 1221
9.121.2.1 total_count . . . 1221
9.121.2.2 total_count_change . . . 1222
9.121.2.3 last_policy_id . . . 1222
9.121.2.4 policies . . . 1222
9.122 DDS_OwnershipQosPolicy Struct Reference . . . 1222
9.122.1 Detailed Description . . . 1223
9.122.2 Usage . . . 1223

9.122.2.1 SHARED ownership . . . 1223
9.122.2.2 EXCLUSIVE ownership . . . 1224
9.122.3 Compatibility . . . 1224
9.122.4 Relationship between registration, liveliness and ownership . . . 1225
9.122.4.1 Ownership Resolution on Redundant Systems . . . 1225
9.122.4.2 Detection of Loss in Topological Connectivity . . . 1226
9.122.4.3 Semantic Difference between unregister_instance and dispose . . . 1227
9.122.5 Member Data Documentation . . . 1227
9.122.5.1 kind . . . 1227
9.123 DDS_OwnershipStrengthQosPolicy Struct Reference . . . 1227
9.123.1 Detailed Description . . . 1228
9.123.2 Member Data Documentation . . . 1228
9.123.2.1 value . . . 1228
9.124 DDS_ParticipantBuiltinTopicData Struct Reference . . . 1229
9.124.1 Detailed Description . . . 1230
9.124.2 Member Data Documentation . . . 1230
9.124.2.1 key . . . 1230
9.124.2.2 user_data . . . 1230
9.124.2.3 property . . . 1230
9.124.2.4 rtps_protocol_version . . . 1230
9.124.2.5 rtps_vendor_id . . . 1231
9.124.2.6 dds_builtin_endpoints . . . 1231
9.124.2.7 available_builtin_endpoints_ext . . . 1232
9.124.2.8 default_unicast_locators . . . 1232
9.124.2.9 product_version . . . 1232
9.124.2.10 participant_name . . . 1232
9.124.2.11 domain_id . . . 1232
9.124.2.12 transport_info . . . 1233
9.124.2.13 reachability_lease_duration . . . 1233
9.124.2.14 partition . . . 1233
9.124.2.15 trust_protection_info . . . 1233
9.124.2.16 trust_algorithm_info . . . 1234
9.124.2.17 partial_configuration . . . 1234
9.125 DDS_ParticipantBuiltinTopicDataSeq Struct Reference . . . 1235
9.125.1 Detailed Description . . . 1235
9.126 DDS_ParticipantTrustAlgorithmInfo Struct Reference . . . 1235
9.126.1 Detailed Description . . . 1235
9.126.2 Member Data Documentation . . . 1235
9.126.2.1 signature . . . 1236

9.126.2.2 key_establishment . . . 1236
9.126.2.3 interceptor . . . 1236
9.127 DDS_ParticipantTrustInterceptorAlgorithmInfo Struct Reference . . . 1236
9.127.1 Detailed Description . . . 1236
9.127.2 Member Data Documentation . . . 1237
9.127.2.1 supported_mask . . . 1237
9.127.2.2 builtin_endpoints_required_mask . . . 1237
9.127.2.3 builtin_kx_endpoints_required_mask . . . 1237
9.127.2.4 user_endpoints_default_required_mask . . . 1237
9.128 DDS_ParticipantTrustKeyEstablishmentAlgorithmInfo Struct Reference . . . 1237
9.128.1 Detailed Description . . . 1238
9.128.2 Member Data Documentation . . . 1238
9.128.2.1 shared_secret . . . 1238
9.129 DDS_ParticipantTrustProtectionInfo Struct Reference . . . 1238
9.129.1 Detailed Description . . . 1238
9.129.2 Member Data Documentation . . . 1238
9.129.2.1 bitmask . . . 1239
9.129.2.2 plugin_bitmask . . . 1239
9.130 DDS_ParticipantTrustSignatureAlgorithmInfo Struct Reference . . . 1239
9.130.1 Detailed Description . . . 1239
9.130.2 Member Data Documentation . . . 1239
9.130.2.1 trust_chain . . . 1240
9.130.2.2 message_auth . . . 1240
9.131 DDS_PartitionQosPolicy Struct Reference . . . 1240
9.131.1 Detailed Description . . . 1240
9.131.2 Usage . . . 1241
9.131.3 Member Data Documentation . . . 1242
9.131.3.1 name . . . 1242
9.132 DDS_PersistentStorageSettings Struct Reference . . . 1242
9.132.1 Detailed Description . . . 1243
9.132.2 Member Data Documentation . . . 1244
9.132.2.1 enable . . . 1244
9.132.2.2 file_name . . . 1244
9.132.2.3 trace_file_name . . . 1245
9.132.2.4 journal_kind . . . 1245
9.132.2.5 synchronization_kind . . . 1245
9.132.2.6 vacuum . . . 1246
9.132.2.7 restore . . . 1246
9.132.2.8 writer_instance_cache_allocation . . . 1247

9.132.2.9 writer_sample_cache_allocation . . . 1247
9.132.2.10 writer_memory_state . . . 1248
9.132.2.11 reader_checkpoint_frequency . . . 1248
9.133 DDS_PresentationQosPolicy Struct Reference . . . 1249
9.133.1 Detailed Description . . . 1249
9.133.2 Usage . . . 1250
9.133.3 Compatibility . . . 1251
9.133.4 Member Data Documentation . . . 1252
9.133.4.1 access_scope . . . 1252
9.133.4.2 coherent_access . . . 1252
9.133.4.3 ordered_access . . . 1253
9.133.4.4 drop_incomplete_coherent_set . . . 1253
9.134 DDS_PrintFormatProperty Struct Reference . . . 1253
9.134.1 Detailed Description . . . 1254
9.134.2 Member Data Documentation . . . 1254
9.134.2.1 kind . . . 1254
9.134.2.2 pretty_print . . . 1254
9.134.2.3 enum_as_int . . . 1255
9.134.2.4 include_root_elements . . . 1255
9.135 DDS_ProductVersion_t Struct Reference . . . 1255
9.135.1 Detailed Description . . . 1256
9.135.2 Member Data Documentation . . . 1256
9.135.2.1 major . . . 1256
9.135.2.2 minor . . . 1256
9.135.2.3 release . . . 1256
9.135.2.4 revision . . . 1257
9.136 DDS_ProfileQosPolicy Struct Reference . . . 1257
9.136.1 Detailed Description . . . 1257
9.136.2 Member Data Documentation . . . 1258
9.136.2.1 string_profile . . . 1258
9.136.2.2 url_profile . . . 1258
9.136.2.3 ignore_user_profile . . . 1258
9.136.2.4 ignore_environment_profile . . . 1259
9.136.2.5 ignore_resource_profile . . . 1259
9.137 DDS_Property_t Struct Reference . . . 1259
9.137.1 Detailed Description . . . 1260
9.137.2 Member Data Documentation . . . 1260
9.137.2.1 name . . . 1260
9.137.2.2 value . . . 1260

9.137.2.3 propagate . . . 1260
9.138 DDS_PropertyQosPolicy Struct Reference . . . 1260
9.138.1 Detailed Description . . . 1261
9.138.2 Usage . . . 1261
9.138.2.1 Reasons for Using the PropertyQosPolicy . . . 1262
9.138.3 Member Data Documentation . . . 1262
9.138.3.1 value . . . 1262
9.139 DDS_PropertySeq Struct Reference . . . 1262
9.139.1 Detailed Description . . . 1263
9.140 DDS_ProtocolVersion_t Struct Reference . . . 1263
9.140.1 Detailed Description . . . 1263
9.140.2 Member Data Documentation . . . 1263
9.140.2.1 major . . . 1263
9.140.2.2 minor . . . 1263
9.141 DDS_PublicationBuiltinTopicData Struct Reference . . . 1264
9.141.1 Detailed Description . . . 1265
9.141.2 Member Data Documentation . . . 1266
9.141.2.1 key . . . 1266
9.141.2.2 participant_key . . . 1266
9.141.2.3 topic_name . . . 1266
9.141.2.4 type_name . . . 1266
9.141.2.5 durability . . . 1267
9.141.2.6 durability_service . . . 1267
9.141.2.7 deadline . . . 1267
9.141.2.8 latency_budget . . . 1267
9.141.2.9 liveliness . . . 1267
9.141.2.10 reliability . . . 1267
9.141.2.11 lifespan . . . 1268
9.141.2.12 user_data . . . 1268
9.141.2.13 ownership . . . 1268
9.141.2.14 ownership_strength . . . 1268
9.141.2.15 destination_order . . . 1268
9.141.2.16 presentation . . . 1269
9.141.2.17 partition . . . 1269
9.141.2.18 topic_data . . . 1269
9.141.2.19 group_data . . . 1269
9.141.2.20 representation . . . 1269
9.141.2.21 data_tags . . . 1269
9.141.2.22 type_code . . . 1270

9.141.2.23 publisher_key . . . 1270
9.141.2.24 property . . . 1270
9.141.2.25 unicast_locators . . . 1270
9.141.2.26 virtual_guid . . . 1271
9.141.2.27 service . . . 1271
9.141.2.28 rtps_protocol_version . . . 1271
9.141.2.29 rtps_vendor_id . . . 1271
9.141.2.30 product_version . . . 1271
9.141.2.31 locator_filter . . . 1272
9.141.2.32 disable_positive_acks . . . 1272
9.141.2.33 publication_name . . . 1272
9.141.2.34 trust_protection_info . . . 1272
9.141.2.35 trust_algorithm_info . . . 1273
9.141.2.36 related_datareader_key . . . 1273
9.142 DDS_PublicationBuiltinTopicDataSeq Struct Reference . . . 1273
9.142.1 Detailed Description . . . 1273
9.143 DDS_PublicationMatchedStatus Struct Reference . . . 1273
9.143.1 Detailed Description . . . 1274
9.143.2 Member Data Documentation . . . 1274
9.143.2.1 total_count . . . 1275
9.143.2.2 total_count_change . . . 1275
9.143.2.3 current_count . . . 1275
9.143.2.4 current_count_peak . . . 1275
9.143.2.5 current_count_change . . . 1275
9.143.2.6 last_subscription_handle . . . 1276
9.144 DDS_PublisherQos Struct Reference . . . 1276
9.144.1 Detailed Description . . . 1277
9.144.2 Member Function Documentation . . . 1277
9.144.2.1 operator==() . . . 1277
9.144.2.2 operator"!=() . . . 1278
9.144.3 Member Data Documentation . . . 1278
9.144.3.1 presentation . . . 1278
9.144.3.2 partition . . . 1278
9.144.3.3 group_data . . . 1278
9.144.3.4 entity_factory . . . 1279
9.144.3.5 asynchronous_publisher . . . 1279
9.144.3.6 publisher_name . . . 1279
9.145 DDS_PublishModeQosPolicy Struct Reference . . . 1279
9.145.1 Detailed Description . . . 1280

9.145.2 Usage . . . 1280
9.145.3 Member Data Documentation . . . 1281
9.145.3.1 kind . . . 1281
9.145.3.2 flow_controller_name . . . 1281
9.145.3.3 priority . . . 1283
9.146 DDS_QosPolicyCount Struct Reference . . . 1283
9.146.1 Detailed Description . . . 1284
9.146.2 Member Data Documentation . . . 1284
9.146.2.1 policy_id . . . 1284
9.146.2.2 count . . . 1284
9.147 DDS_QosPolicyCountSeq Struct Reference . . . 1284
9.147.1 Detailed Description . . . 1284
9.148 DDS_QosPrintAll_t Struct Reference . . . 1285
9.148.1 Detailed Description . . . 1285
9.149 DDS_QosPrintFormat Struct Reference . . . 1285
9.149.1 Detailed Description . . . 1285
9.149.2 Member Data Documentation . . . 1285
9.149.2.1 is_standalone . . . 1286
9.149.2.2 print_private . . . 1286
9.149.2.3 indent . . . 1287
9.150 DDS_QueryConditionParams Struct Reference . . . 1287
9.150.1 Detailed Description . . . 1288
9.150.2 Member Data Documentation . . . 1288
9.150.2.1 as_readconditionparams . . . 1288
9.150.2.2 query_expression . . . 1288
9.150.2.3 query_parameters . . . 1288
9.151 DDS_ReadConditionParams Struct Reference . . . 1288
9.151.1 Detailed Description . . . 1289
9.151.2 Member Data Documentation . . . 1289
9.151.2.1 sample_states . . . 1289
9.151.2.2 view_states . . . 1289
9.151.2.3 instance_states . . . 1289
9.151.2.4 stream_kinds . . . 1289
9.152 DDS_ReaderDataLifecycleQosPolicy Struct Reference . . . 1289
9.152.1 Detailed Description . . . 1290
9.152.2 Member Data Documentation . . . 1291
9.152.2.1 autopurge_nowriter_samples_delay . . . 1291
9.152.2.2 autopurge_disposed_samples_delay . . . 1291
9.152.2.3 autopurge_disposed_instances_delay . . . 1292

9.152.2.4 autopurge_nowriter_instances_delay . . . 1292
9.153 DDS_ReceiverPoolQosPolicy Struct Reference . . . 1293
9.153.1 Detailed Description . . . 1293
9.153.2 Usage . . . 1293
9.153.3 Member Data Documentation . . . 1294
9.153.3.1 thread . . . 1294
9.153.3.2 buffer_size . . . 1294
9.153.3.3 buffer_alignment . . . 1295
9.154 DDS_ReliabilityQosPolicy Struct Reference . . . 1295
9.154.1 Detailed Description . . . 1296
9.154.2 Usage . . . 1296
9.154.3 Compatibility . . . 1297
9.154.4 Member Data Documentation . . . 1298
9.154.4.1 kind . . . 1298
9.154.4.2 max_blocking_time . . . 1298
9.154.4.3 acknowledgment_kind . . . 1299
9.154.4.4 instance_state_consistency_kind . . . 1299
9.154.4.5 exact_reliability_kind_match . . . 1300
9.155 DDS_ReliableReaderActivityChangedStatus Struct Reference . . . 1300
9.155.1 Detailed Description . . . 1301
9.155.2 Member Data Documentation . . . 1301
9.155.2.1 active_count . . . 1301
9.155.2.2 inactive_count . . . 1301
9.155.2.3 active_count_change . . . 1302
9.155.2.4 inactive_count_change . . . 1302
9.155.2.5 last_instance_handle . . . 1302
9.156 DDS_ReliableWriterCacheChangedStatus Struct Reference . . . 1302
9.156.1 Detailed Description . . . 1303
9.156.2 Member Data Documentation . . . 1303
9.156.2.1 empty_reliable_writer_cache . . . 1303
9.156.2.2 full_reliable_writer_cache . . . 1303
9.156.2.3 low_watermark_reliable_writer_cache . . . 1304
9.156.2.4 high_watermark_reliable_writer_cache . . . 1304
9.156.2.5 unacknowledged_sample_count . . . 1304
9.156.2.6 unacknowledged_sample_count_peak . . . 1304
9.156.2.7 replaced_unacknowledged_sample_count . . . 1305
9.157 DDS_ReliableWriterCacheEventCount Struct Reference . . . 1305
9.157.1 Detailed Description . . . 1305
9.157.2 Member Data Documentation . . . 1305

9.157.2.1 total_count . . . 1305
9.157.2.2 total_count_change . . . 1306
9.158 DDS_RequestedDeadlineMissedStatus Struct Reference . . . 1306
9.158.1 Detailed Description . . . 1306
9.158.2 Member Data Documentation . . . 1306
9.158.2.1 total_count . . . 1306
9.158.2.2 total_count_change . . . 1306
9.158.2.3 last_instance_handle . . . 1307
9.159 DDS_RequestedIncompatibleQosStatus Struct Reference . . . 1307
9.159.1 Detailed Description . . . 1307
9.159.2 Member Data Documentation . . . 1307
9.159.2.1 total_count . . . 1308
9.159.2.2 total_count_change . . . 1308
9.159.2.3 last_policy_id . . . 1308
9.159.2.4 policies . . . 1308
9.160 DDS_ResourceLimitsQosPolicy Struct Reference . . . 1308
9.160.1 Detailed Description . . . 1309
9.160.2 Usage . . . 1310
9.160.3 Consistency . . . 1310
9.160.4 Member Data Documentation . . . 1311
9.160.4.1 max_samples . . . 1311
9.160.4.2 max_instances . . . 1311
9.160.4.3 max_samples_per_instance . . . 1312
9.160.4.4 initial_samples . . . 1312
9.160.4.5 initial_instances . . . 1313
9.160.4.6 instance_hash_buckets . . . 1313
9.161 DDS_RTPS_EntityId_t Struct Reference . . . 1313
9.161.1 Detailed Description . . . 1313
9.162 DDS_RTPS_GUID_t Struct Reference . . . 1314
9.162.1 Detailed Description . . . 1314
9.163 DDS_RtpsReliableReaderProtocol_t Struct Reference . . . 1314
9.163.1 Detailed Description . . . 1315
9.163.2 Member Data Documentation . . . 1315
9.163.2.1 min_heartbeat_response_delay . . . 1315
9.163.2.2 max_heartbeat_response_delay . . . 1316
9.163.2.3 heartbeat_suppression_duration . . . 1316
9.163.2.4 nack_period . . . 1317
9.163.2.5 receive_window_size . . . 1317
9.163.2.6 round_trip_time . . . 1318

9.163.2.7 app_ack_period . . . 1318
9.163.2.8 min_app_ack_response_keep_duration . . . 1319
9.163.2.9 samples_per_app_ack . . . 1319
9.164 DDS_RtpsReliableWriterProtocol_t Struct Reference . . . 1320
9.164.1 Detailed Description . . . 1321
9.164.2 Member Data Documentation . . . 1321
9.164.2.1 low_watermark . . . 1322
9.164.2.2 high_watermark . . . 1322
9.164.2.3 heartbeat_period . . . 1323
9.164.2.4 fast_heartbeat_period . . . 1324
9.164.2.5 late_joiner_heartbeat_period . . . 1324
9.164.2.6 virtual_heartbeat_period . . . 1325
9.164.2.7 samples_per_virtual_heartbeat . . . 1326
9.164.2.8 max_heartbeat_retries . . . 1326
9.164.2.9 inactivate_nonprogressing_readers . . . 1327
9.164.2.10 heartbeats_per_max_samples . . . 1327
9.164.2.11 min_nack_response_delay . . . 1328
9.164.2.12 max_nack_response_delay . . . 1329
9.164.2.13 nack_suppression_duration . . . 1329
9.164.2.14 max_bytes_per_nack_response . . . 1330
9.164.2.15 disable_positive_acks_min_sample_keep_duration . . . 1331
9.164.2.16 disable_positive_acks_max_sample_keep_duration . . . 1331
9.164.2.17 disable_positive_acks_enable_adaptive_sample_keep_duration . . . 1332
9.164.2.18 disable_positive_acks_decrease_sample_keep_duration_factor . . . 1332
9.164.2.19 disable_positive_acks_increase_sample_keep_duration_factor . . . 1333
9.164.2.20 min_send_window_size . . . 1333
9.164.2.21 max_send_window_size . . . 1334
9.164.2.22 send_window_update_period . . . 1335
9.164.2.23 send_window_increase_factor . . . 1336
9.164.2.24 send_window_decrease_factor . . . 1336
9.164.2.25 enable_multicast_periodic_heartbeat . . . 1337
9.164.2.26 multicast_resend_threshold . . . 1338
9.164.2.27 disable_repair_piggyback_heartbeat . . . 1338
9.165 DDS_RtpsWellKnownPorts_t Struct Reference . . . 1338
9.165.1 Detailed Description . . . 1339
9.165.2 Member Data Documentation . . . 1340
9.165.2.1 port_base . . . 1341
9.165.2.2 domain_id_gain . . . 1341
9.165.2.3 participant_id_gain . . . 1342

9.165.2.4 builtin_multicast_port_offset . . . 1342
9.165.2.5 builtin_unicast_port_offset . . . 1343
9.165.2.6 user_multicast_port_offset . . . 1343
9.165.2.7 user_unicast_port_offset . . . 1343
9.166 DDS_SampleIdentity_t Struct Reference . . . 1343
9.166.1 Detailed Description . . . 1344
9.167 DDS_SampleInfo Struct Reference . . . 1344
9.167.1 Detailed Description . . . 1345
9.167.2 Interpretation of the SampleInfo . . . 1345
9.167.3 Interpretation of the SampleInfo disposed_generation_count and no_writers_generation_count 1346
9.167.4 Interpretation of the SampleInfo sample_rank, generation_rank and absolute_generation_rank 1346
9.167.5 Interpretation of the SampleInfo counters and ranks . . . 1347
9.167.6 Member Data Documentation . . . 1347
9.167.6.1 sample_state . . . 1348
9.167.6.2 view_state . . . 1348
9.167.6.3 instance_state . . . 1348
9.167.6.4 source_timestamp . . . 1349
9.167.6.5 instance_handle . . . 1349
9.167.6.6 publication_handle . . . 1349
9.167.6.7 disposed_generation_count . . . 1349
9.167.6.8 no_writers_generation_count . . . 1350
9.167.6.9 sample_rank . . . 1350
9.167.6.10 generation_rank . . . 1350
9.167.6.11 absolute_generation_rank . . . 1351
9.167.6.12 valid_data . . . 1351
9.167.6.13 reception_timestamp . . . 1351
9.167.6.14 publication_sequence_number . . . 1352
9.167.6.15 reception_sequence_number . . . 1352
9.167.6.16 original_publication_virtual_guid . . . 1352
9.167.6.17 original_publication_virtual_sequence_number . . . 1352
9.167.6.18 related_original_publication_virtual_guid . . . 1353
9.167.6.19 related_original_publication_virtual_sequence_number . . . 1353
9.167.6.20 flag . . . 1353
9.167.6.21 source_guid . . . 1353
9.167.6.22 related_source_guid . . . 1354
9.167.6.23 related_subscription_guid . . . 1354
9.167.6.24 topic_query_guid . . . 1354
9.167.6.25 coherent_set_info . . . 1354
9.168 DDS_SampleInfoSeq Struct Reference . . . 1355

9.168.1 Detailed Description . . . 1355
9.169 DDS_SampleLostStatus Struct Reference . . . 1355
9.169.1 Detailed Description . . . 1355
9.169.2 Member Data Documentation . . . 1355
9.169.2.1 total_count . . . 1355
9.169.2.2 total_count_change . . . 1356
9.169.2.3 last_reason . . . 1356
9.170 DDS_SampleRejectedStatus Struct Reference . . . 1356
9.170.1 Detailed Description . . . 1356
9.170.2 Member Data Documentation . . . 1356
9.170.2.1 total_count . . . 1357
9.170.2.2 total_count_change . . . 1357
9.170.2.3 last_reason . . . 1357
9.170.2.4 last_instance_handle . . . 1357
9.171 DDS_SequenceNumber_t Struct Reference . . . 1357
9.171.1 Detailed Description . . . 1358
9.171.2 Member Data Documentation . . . 1358
9.171.2.1 high . . . 1358
9.171.2.2 low . . . 1358
9.172 DDS_ServiceQosPolicy Struct Reference . . . 1358
9.172.1 Detailed Description . . . 1359
9.172.2 Member Data Documentation . . . 1359
9.172.2.1 kind . . . 1359
9.173 DDS_ServiceRequest Struct Reference . . . 1359
9.173.1 Detailed Description . . . 1360
9.173.2 Member Data Documentation . . . 1360
9.173.2.1 service_id . . . 1360
9.173.2.2 instance_id . . . 1360
9.173.2.3 request_body . . . 1361
9.174 DDS_ServiceRequestAcceptedStatus Struct Reference . . . 1361
9.174.1 Detailed Description . . . 1361
9.174.2 Member Data Documentation . . . 1362
9.174.2.1 total_count . . . 1362
9.174.2.2 total_count_change . . . 1362
9.174.2.3 current_count . . . 1362
9.174.2.4 current_count_change . . . 1362
9.174.2.5 last_request_handle . . . 1362
9.174.2.6 service_id . . . 1363
9.175 DDS_ServiceRequestSeq Struct Reference . . . 1363

9.175.1 Detailed Description . . . 1363
9.176 DDS_ShortSeq Struct Reference . . . 1363
9.176.1 Detailed Description . . . 1363
9.177 DDS_StringSeq Struct Reference . . . 1364
9.177.1 Detailed Description . . . 1364
9.178 DDS_StructMember Struct Reference . . . 1365
9.178.1 Detailed Description . . . 1365
9.178.2 Member Data Documentation . . . 1365
9.178.2.1 name . . . 1365
9.178.2.2 type . . . 1366
9.178.2.3 is_pointer . . . 1366
9.178.2.4 bits . . . 1366
9.178.2.5 is_key . . . 1366
9.178.2.6 id . . . 1366
9.178.2.7 is_optional . . . 1366
9.178.2.8 is_must_understand . . . 1367
9.179 DDS_StructMemberSeq Struct Reference . . . 1367
9.179.1 Detailed Description . . . 1367
9.180 DDS_SubscriberQos Struct Reference . . . 1367
9.180.1 Detailed Description . . . 1368
9.180.2 Member Function Documentation . . . 1368
9.180.2.1 operator==() . . . 1368
9.180.2.2 operator"!=() . . . 1369
9.180.3 Member Data Documentation . . . 1369
9.180.3.1 presentation . . . 1369
9.180.3.2 partition . . . 1369
9.180.3.3 group_data . . . 1369
9.180.3.4 entity_factory . . . 1370
9.180.3.5 subscriber_name . . . 1370
9.181 DDS_SubscriptionBuiltinTopicData Struct Reference . . . 1370
9.181.1 Detailed Description . . . 1372
9.181.2 Member Data Documentation . . . 1372
9.181.2.1 key . . . 1372
9.181.2.2 participant_key . . . 1372
9.181.2.3 topic_name . . . 1372
9.181.2.4 type_name . . . 1373
9.181.2.5 durability . . . 1373
9.181.2.6 deadline . . . 1373
9.181.2.7 latency_budget . . . 1373

9.181.2.8 liveliness . . . 1373
9.181.2.9 reliability . . . 1374
9.181.2.10 ownership . . . 1374
9.181.2.11 destination_order . . . 1374
9.181.2.12 user_data . . . 1374
9.181.2.13 time_based_filter . . . 1374
9.181.2.14 presentation . . . 1375
9.181.2.15 partition . . . 1375
9.181.2.16 topic_data . . . 1375
9.181.2.17 group_data . . . 1375
9.181.2.18 type_consistency . . . 1375
9.181.2.19 representation . . . 1375
9.181.2.20 data_tags . . . 1376
9.181.2.21 type_code . . . 1376
9.181.2.22 subscriber_key . . . 1376
9.181.2.23 property . . . 1376
9.181.2.24 unicast_locators . . . 1376
9.181.2.25 multicast_locators . . . 1377
9.181.2.26 content_filter_property . . . 1377
9.181.2.27 virtual_guid . . . 1377
9.181.2.28 service . . . 1377
9.181.2.29 rtps_protocol_version . . . 1377
9.181.2.30 rtps_vendor_id . . . 1378
9.181.2.31 product_version . . . 1378
9.181.2.32 disable_positive_acks . . . 1378
9.181.2.33 subscription_name . . . 1378
9.181.2.34 trust_protection_info . . . 1378
9.181.2.35 trust_algorithm_info . . . 1379
9.181.2.36 related_datawriter_key . . . 1379
9.182 DDS_SubscriptionBuiltinTopicDataSeq Struct Reference . . . 1379
9.182.1 Detailed Description . . . 1379
9.183 DDS_SubscriptionMatchedStatus Struct Reference . . . 1379
9.183.1 Detailed Description . . . 1380
9.183.2 Member Data Documentation . . . 1380
9.183.2.1 total_count . . . 1381
9.183.2.2 total_count_change . . . 1381
9.183.2.3 current_count . . . 1381
9.183.2.4 current_count_peak . . . 1381
9.183.2.5 current_count_change . . . 1381

9.183.2.6 last_publication_handle . . . 1382
9.184 DDS_SystemResourceLimitsQosPolicy Struct Reference . . . 1382
9.184.1 Detailed Description . . . 1382
9.184.2 Usage . . . 1383
9.184.3 Member Data Documentation . . . 1383
9.184.3.1 max_objects_per_thread . . . 1383
9.184.3.2 initial_objects_per_thread . . . 1384
9.185 DDS_Tag Struct Reference . . . 1384
9.185.1 Detailed Description . . . 1384
9.185.2 Member Data Documentation . . . 1384
9.185.2.1 name . . . 1385
9.185.2.2 value . . . 1385
9.186 DDS_TagSeq Struct Reference . . . 1385
9.186.1 Detailed Description . . . 1385
9.187 DDS_ThreadSettings_t Struct Reference . . . 1385
9.187.1 Detailed Description . . . 1386
9.187.2 Member Data Documentation . . . 1386
9.187.2.1 mask . . . 1386
9.187.2.2 priority . . . 1386
9.187.2.3 stack_size . . . 1386
9.187.2.4 cpu_list . . . 1387
9.187.2.5 cpu_rotation . . . 1387
9.188 DDS_Time_t Struct Reference . . . 1387
9.188.1 Detailed Description . . . 1388
9.188.2 Member Data Documentation . . . 1388
9.188.2.1 sec . . . 1388
9.188.2.2 nanosec . . . 1388
9.189 DDS_TimeBasedFilterQosPolicy Struct Reference . . . 1389
9.189.1 Detailed Description . . . 1389
9.189.2 Usage . . . 1389
9.189.3 Consistency . . . 1390
9.189.4 Member Data Documentation . . . 1391
9.189.4.1 minimum_separation . . . 1391
9.190 DDS_TopicBuiltinTopicData Struct Reference . . . 1391
9.190.1 Detailed Description . . . 1392
9.190.2 Member Data Documentation . . . 1392
9.190.2.1 key . . . 1392
9.190.2.2 name . . . 1393
9.190.2.3 type_name . . . 1393

9.190.2.4 durability . . . 1393
9.190.2.5 durability_service . . . 1393
9.190.2.6 deadline . . . 1394
9.190.2.7 latency_budget . . . 1394
9.190.2.8 liveliness . . . 1394
9.190.2.9 reliability . . . 1394
9.190.2.10 transport_priority . . . 1394
9.190.2.11 lifespan . . . 1394
9.190.2.12 destination_order . . . 1395
9.190.2.13 history . . . 1395
9.190.2.14 resource_limits . . . 1395
9.190.2.15 ownership . . . 1395
9.190.2.16 topic_data . . . 1395
9.190.2.17 representation . . . 1395
9.191 DDS_TopicBuiltinTopicDataSeq Struct Reference . . . 1396
9.191.1 Detailed Description . . . 1396
9.192 DDS_TopicDataQosPolicy Struct Reference . . . 1396
9.192.1 Detailed Description . . . 1396
9.192.2 Usage . . . 1397
9.192.3 Member Data Documentation . . . 1397
9.192.3.1 value . . . 1397
9.193 DDS_TopicQos Struct Reference . . . 1397
9.193.1 Detailed Description . . . 1399
9.193.2 Member Function Documentation . . . 1399
9.193.2.1 operator==() . . . 1399
9.193.2.2 operator"!=() . . . 1400
9.193.3 Member Data Documentation . . . 1400
9.193.3.1 topic_data . . . 1400
9.193.3.2 durability . . . 1400
9.193.3.3 durability_service . . . 1400
9.193.3.4 deadline . . . 1401
9.193.3.5 latency_budget . . . 1401
9.193.3.6 liveliness . . . 1401
9.193.3.7 reliability . . . 1401
9.193.3.8 destination_order . . . 1401
9.193.3.9 history . . . 1401
9.193.3.10 resource_limits . . . 1402
9.193.3.11 transport_priority . . . 1402
9.193.3.12 lifespan . . . 1402

9.193.3.13 ownership . . . 1402
9.193.3.14 representation . . . 1402
9.194 DDS_TopicQueryData Struct Reference . . . 1402
9.194.1 Detailed Description . . . 1403
9.194.2 Member Data Documentation . . . 1403
9.194.2.1 topic_query_selection . . . 1403
9.194.2.2 topic_name . . . 1403
9.194.2.3 original_related_reader_guid . . . 1403
9.195 DDS_TopicQueryDispatchQosPolicy Struct Reference . . . 1404
9.195.1 Detailed Description . . . 1404
9.195.2 Member Data Documentation . . . 1405
9.195.2.1 enable . . . 1405
9.195.2.2 publication_period . . . 1405
9.195.2.3 samples_per_period . . . 1406
9.196 DDS_TopicQuerySelection Struct Reference . . . 1406
9.196.1 Detailed Description . . . 1406
9.196.2 Member Data Documentation . . . 1407
9.196.2.1 filter_class_name . . . 1407
9.196.2.2 filter_expression . . . 1407
9.196.2.3 filter_parameters . . . 1407
9.196.2.4 kind . . . 1408
9.197 DDS_TransportBuiltinQosPolicy Struct Reference . . . 1408
9.197.1 Detailed Description . . . 1408
9.197.2 Member Data Documentation . . . 1408
9.197.2.1 mask . . . 1409
9.198 DDS_TransportInfo_t Struct Reference . . . 1409
9.198.1 Detailed Description . . . 1409
9.198.2 Member Data Documentation . . . 1409
9.198.2.1 class_id . . . 1409
9.198.2.2 message_size_max . . . 1410
9.199 DDS_TransportInfoSeq Struct Reference . . . 1410
9.199.1 Detailed Description . . . 1410
9.200 DDS_TransportMulticastMapping_t Struct Reference . . . 1410
9.200.1 Detailed Description . . . 1411
9.200.2 Member Data Documentation . . . 1411
9.200.2.1 addresses . . . 1411
9.200.2.2 topic_expression . . . 1412
9.200.2.3 mapping_function . . . 1412
9.201 DDS_TransportMulticastMappingFunction_t Struct Reference . . . 1412

9.201.1 Detailed Description . . . 1412
9.201.2 Member Data Documentation . . . 1412
9.201.2.1 dll . . . 1413
9.201.2.2 function_name . . . 1413
9.202 DDS_TransportMulticastMappingQosPolicy Struct Reference . . . 1413
9.202.1 Detailed Description . . . 1414
9.202.2 Member Data Documentation . . . 1415
9.202.2.1 value . . . 1415
9.203 DDS_TransportMulticastMappingSeq Struct Reference . . . 1415
9.203.1 Detailed Description . . . 1415
9.204 DDS_TransportMulticastQosPolicy Struct Reference . . . 1415
9.204.1 Detailed Description . . . 1416
9.204.2 Member Data Documentation . . . 1416
9.204.2.1 value . . . 1416
9.204.2.2 kind . . . 1417
9.205 DDS_TransportMulticastSettings_t Struct Reference . . . 1417
9.205.1 Detailed Description . . . 1417
9.205.2 Member Data Documentation . . . 1418
9.205.2.1 transports . . . 1418
9.205.2.2 receive_address . . . 1418
9.205.2.3 receive_port . . . 1419
9.206 DDS_TransportMulticastSettingsSeq Struct Reference . . . 1419
9.206.1 Detailed Description . . . 1419
9.207 DDS_TransportPriorityQosPolicy Struct Reference . . . 1419
9.207.1 Detailed Description . . . 1420
9.207.2 Usage . . . 1420
9.207.3 Member Data Documentation . . . 1421
9.207.3.1 value . . . 1421
9.208 DDS_TransportSelectionQosPolicy Struct Reference . . . 1421
9.208.1 Detailed Description . . . 1421
9.208.2 Member Data Documentation . . . 1422
9.208.2.1 enabled_transports . . . 1422
9.209 DDS_TransportUnicastQosPolicy Struct Reference . . . 1422
9.209.1 Detailed Description . . . 1423
9.209.2 Usage . . . 1423
9.209.3 Member Data Documentation . . . 1424
9.209.3.1 value . . . 1424
9.210 DDS_TransportUnicastSettings_t Struct Reference . . . 1424
9.210.1 Detailed Description . . . 1425

9.210.2 Member Data Documentation . . . 1425
9.210.2.1 transports . . . 1425
9.210.2.2 receive_port . . . 1425
9.211 DDS_TransportUnicastSettingsSeq Struct Reference . . . 1426
9.211.1 Detailed Description . . . 1426
9.212 DDS_TrustAlgorithmRequirements Struct Reference . . . 1426
9.212.1 Detailed Description . . . 1426
9.212.2 Member Data Documentation . . . 1426
9.212.2.1 supported_mask . . . 1426
9.212.2.2 required_mask . . . 1427
9.213 DDS_TypeAllocationParams_t Struct Reference . . . 1427
9.213.1 Detailed Description . . . 1427
9.213.2 Member Function Documentation . . . 1427
9.213.2.1 set_allocate_pointers() . . . 1428
9.213.2.2 set_allocate_optional_members() . . . 1428
9.213.3 Member Data Documentation . . . 1428
9.213.3.1 allocate_pointers . . . 1428
9.213.3.2 allocate_optional_members . . . 1428
9.214 DDS_TypeCode Struct Reference . . . 1428
9.214.1 Detailed Description . . . 1431
9.214.2 Member Function Documentation . . . 1432
9.214.2.1 kind() . . . 1432
9.214.2.2 extensibility_kind() . . . 1432
9.214.2.3 equal() . . . 1435
9.214.2.4 is_assignable_from() . . . 1436
9.214.2.5 name() . . . 1436
9.214.2.6 member_count() . . . 1437
9.214.2.7 member_name() . . . 1438
9.214.2.8 find_member_by_name() . . . 1439
9.214.2.9 member_type() . . . 1439
9.214.2.10 member_label_count() . . . 1440
9.214.2.11 member_label() . . . 1441
9.214.2.12 member_ordinal() . . . 1442
9.214.2.13 is_member_key() . . . 1443
9.214.2.14 is_member_required() . . . 1444
9.214.2.15 is_member_pointer() . . . 1444
9.214.2.16 is_member_bitfield() . . . 1445
9.214.2.17 member_bitfield_bits() . . . 1446
9.214.2.18 member_visibility() . . . 1446

9.214.2.19 discriminator_type() . . . 1447
9.214.2.20 length() . . . 1448
9.214.2.21 array_dimension_count() . . . 1449
9.214.2.22 array_dimension() . . . 1450
9.214.2.23 element_count() . . . 1451
9.214.2.24 content_type() . . . 1451
9.214.2.25 is_alias_pointer() . . . 1452
9.214.2.26 default_index() . . . 1452
9.214.2.27 concrete_base_type() . . . 1453
9.214.2.28 type_modifier() . . . 1454
9.214.2.29 member_id() . . . 1455
9.214.2.30 find_member_by_id() . . . 1456
9.214.2.31 get_type_object_serialized_size() . . . 1457
9.214.2.32 get_cdr_serialized_sample_max_size() . . . 1457
9.214.2.33 cdr_serialized_sample_max_size() [1/2] . . . 1458
9.214.2.34 cdr_serialized_sample_min_size() [1/2] . . . 1458
9.214.2.35 cdr_serialized_sample_key_max_size() [1/2] . . . 1459
9.214.2.36 cdr_serialized_sample_max_size() [2/2] . . . 1459
9.214.2.37 cdr_serialized_sample_min_size() [2/2] . . . 1460
9.214.2.38 cdr_serialized_sample_key_max_size() [2/2] . . . 1461
9.214.2.39 add_member_to_enum() . . . 1461
9.214.2.40 add_member_to_union() . . . 1462
9.214.2.41 add_member() . . . 1463
9.214.2.42 add_member_ex() . . . 1465
9.214.2.43 default_value() . . . 1467
9.214.2.44 min_value() . . . 1467
9.214.2.45 max_value() . . . 1468
9.214.2.46 unit() . . . 1469
9.214.2.47 member_default_value() . . . 1470
9.214.2.48 member_min_value() . . . 1470
9.214.2.49 member_max_value() . . . 1471
9.214.2.50 member_unit() . . . 1472
9.214.2.51 is_member_must_understand() . . . 1473
9.214.2.52 print_IDL() . . . 1474
9.214.2.53 print() . . . 1474
9.214.2.54 to_string() [1/2] . . . 1475
9.214.2.55 to_string() [2/2] . . . 1476
9.215 DDS_TypeCodeFactory Struct Reference . . . 1476
9.215.1 Detailed Description . . . 1478

9.215.2 Member Function Documentation . . . 1479
9.215.2.1 get_instance() . . . 1479
9.215.2.2 clone_tc() . . . 1479
9.215.2.3 delete_tc() . . . 1480
9.215.2.4 get_primitive_tc() . . . 1480
9.215.2.5 create_struct_tc() [1/2] . . . 1481
9.215.2.6 create_struct_tc() [2/2] . . . 1481
9.215.2.7 create_value_tc() [1/2] . . . 1482
9.215.2.8 create_value_tc() [2/2] . . . 1483
9.215.2.9 create_union_tc() [1/2] . . . 1483
9.215.2.10 create_union_tc() [2/2] . . . 1484
9.215.2.11 create_enum_tc() [1/2] . . . 1485
9.215.2.12 create_enum_tc() [2/2] . . . 1486
9.215.2.13 create_alias_tc() . . . 1486
9.215.2.14 create_string_tc() . . . 1487
9.215.2.15 create_wstring_tc() . . . 1488
9.215.2.16 create_sequence_tc() . . . 1488
9.215.2.17 create_array_tc() [1/2] . . . 1489
9.215.2.18 create_array_tc() [2/2] . . . 1489
9.216 DDS_TypeCodePrintFormatProperty Struct Reference . . . 1490
9.216.1 Detailed Description . . . 1490
9.216.2 Member Data Documentation . . . 1490
9.216.2.1 indent . . . 1491
9.216.2.2 print_ordinals . . . 1491
9.216.2.3 print_kind . . . 1492
9.216.2.4 print_complete_type . . . 1492
9.217 DDS_TypeConsistencyEnforcementQosPolicy Struct Reference . . . 1493
9.217.1 Detailed Description . . . 1494
9.217.2 Member Data Documentation . . . 1494
9.217.2.1 kind . . . 1495
9.217.2.2 ignore_sequence_bounds . . . 1495
9.217.2.3 ignore_string_bounds . . . 1495
9.217.2.4 ignore_member_names . . . 1496
9.217.2.5 prevent_type_widening . . . 1496
9.217.2.6 force_type_validation . . . 1496
9.217.2.7 ignore_enum_literal_names . . . 1497
9.218 DDS_TypeDeallocationParams_t Struct Reference . . . 1497
9.218.1 Detailed Description . . . 1497
9.218.2 Member Function Documentation . . . 1498

9.218.2.1 set_delete_pointers() . . . 1498
9.218.2.2 set_delete_optional_members() . . . 1498
9.218.3 Member Data Documentation . . . 1498
9.218.3.1 delete_pointers . . . 1498
9.218.3.2 delete_optional_members . . . 1498
9.219 DDS_TypeSupportQosPolicy Struct Reference . . . 1499
9.219.1 Detailed Description . . . 1499
9.219.2 Usage . . . 1499
9.219.3 Member Data Documentation . . . 1500
9.219.3.1 plugin_data . . . 1500
9.219.3.2 cdr_padding_kind . . . 1500
9.220 DDS_UInt8Seq Struct Reference . . . 1500
9.220.1 Detailed Description . . . 1501
9.221 DDS_UnionMember Struct Reference . . . 1501
9.221.1 Detailed Description . . . 1501
9.221.2 Member Data Documentation . . . 1501
9.221.2.1 name . . . 1502
9.221.2.2 is_pointer . . . 1502
9.221.2.3 labels . . . 1502
9.221.2.4 type . . . 1502
9.222 DDS_UnionMemberSeq Struct Reference . . . 1502
9.222.1 Detailed Description . . . 1502
9.223 DDS_UnsignedLongLongSeq Struct Reference . . . 1503
9.223.1 Detailed Description . . . 1503
9.224 DDS_UnsignedLongSeq Struct Reference . . . 1503
9.224.1 Detailed Description . . . 1503
9.225 DDS_UnsignedShortSeq Struct Reference . . . 1503
9.225.1 Detailed Description . . . 1504
9.226 DDS_UserDataQosPolicy Struct Reference . . . 1504
9.226.1 Detailed Description . . . 1504
9.226.2 Usage . . . 1505
9.226.3 Member Data Documentation . . . 1505
9.226.3.1 value . . . 1505
9.227 DDS_ValueMember Struct Reference . . . 1505
9.227.1 Detailed Description . . . 1506
9.227.2 Member Data Documentation . . . 1506
9.227.2.1 name . . . 1506
9.227.2.2 type . . . 1507
9.227.2.3 is_pointer . . . 1507

9.227.2.4 bits . . . 1507
9.227.2.5 is_key . . . 1507
9.227.2.6 access . . . 1507
9.227.2.7 id . . . 1507
9.227.2.8 is_optional . . . 1508
9.227.2.9 is_must_understand . . . 1508
9.228 DDS_ValueMemberSeq Struct Reference . . . 1508
9.228.1 Detailed Description . . . 1508
9.229 DDS_VendorId_t Struct Reference . . . 1508
9.229.1 Detailed Description . . . 1508
9.229.2 Member Data Documentation . . . 1509
9.229.2.1 vendorId . . . 1509
9.230 DDS_VirtualWriterInfo Struct Reference . . . 1509
9.230.1 Detailed Description . . . 1509
9.231 DDS_WaitSetProperty_t Struct Reference . . . 1509
9.231.1 Detailed Description . . . 1509
9.231.2 Member Data Documentation . . . 1510
9.231.2.1 max_event_count . . . 1510
9.231.2.2 max_event_delay . . . 1510
9.232 DDS_WcharSeq Struct Reference . . . 1510
9.232.1 Detailed Description . . . 1511
9.233 DDS_WireProtocolQosPolicy Struct Reference . . . 1511
9.233.1 Detailed Description . . . 1512
9.233.2 Usage . . . 1512
9.233.3 Member Data Documentation . . . 1514
9.233.3.1 participant_id . . . 1515
9.233.3.2 rtps_host_id . . . 1516
9.233.3.3 rtps_app_id . . . 1516
9.233.3.4 rtps_instance_id . . . 1517
9.233.3.5 rtps_well_known_ports . . . 1517
9.233.3.6 rtps_reserved_port_mask . . . 1518
9.233.3.7 rtps_auto_id_kind . . . 1518
9.233.3.8 compute_crc . . . 1518
9.233.3.9 check_crc . . . 1519
9.234 DDS_WriteParams_t Struct Reference . . . 1519
9.234.1 Detailed Description . . . 1520
9.234.2 Member Data Documentation . . . 1520
9.234.2.1 replace_auto . . . 1520
9.234.2.2 identity . . . 1520

9.234.2.3 related_sample_identity . . . 1521
9.234.2.4 source_timestamp . . . 1521
9.234.2.5 cookie . . . 1521
9.234.2.6 handle . . . 1521
9.234.2.7 priority . . . 1522
9.234.2.8 flag . . . 1522
9.234.2.9 source_guid . . . 1523
9.234.2.10 related_source_guid . . . 1523
9.234.2.11 related_reader_guid . . . 1524
9.235 DDS_WriterDataLifecycleQosPolicy Struct Reference . . . 1524
9.235.1 Detailed Description . . . 1524
9.235.2 Usage . . . 1525
9.235.3 Member Data Documentation . . . 1525
9.235.3.1 autodispose_unregistered_instances . . . 1525
9.235.3.2 autopurge_unregistered_instances_delay . . . 1526
9.235.3.3 autopurge_disposed_instances_delay . . . 1527
9.236 DDS_WstringSeq Struct Reference . . . 1527
9.236.1 Detailed Description . . . 1527
9.237 DDSAsyncWaitSet Class Reference . . . 1528
9.237.1 Detailed Description . . . 1529
9.237.2 AsyncWaitSet Thread Orchestration . . . 1529
9.237.3 AsyncWaitSet Thread Safety . . . 1530
9.237.3.1 Condition Locking . . . 1531
9.237.4 AsyncWaitSet Events and Resources . . . 1531
9.237.5 Constructor & Destructor Documentation . . . 1531
9.237.5.1 DDSAsyncWaitSet() [1/4] . . . 1532
9.237.5.2 DDSAsyncWaitSet() [2/4] . . . 1532
9.237.5.3 DDSAsyncWaitSet() [3/4] . . . 1532
9.237.5.4 DDSAsyncWaitSet() [4/4] . . . 1533
9.237.5.5 ∼DDSAsyncWaitSet() . . . 1533
9.237.6 Member Function Documentation . . . 1533
9.237.6.1 start() . . . 1533
9.237.6.2 start_with_completion_token() . . . 1534
9.237.6.3 stop() . . . 1535
9.237.6.4 stop_with_completion_token() . . . 1535
9.237.6.5 attach_condition() . . . 1536
9.237.6.6 detach_condition() . . . 1537
9.237.6.7 attach_condition_with_completion_token() . . . 1537
9.237.6.8 detach_condition_with_completion_token() . . . 1538

9.237.6.9 unlock_condition() . . . 1539
9.237.6.10 get_property() . . . 1539
9.237.6.11 get_conditions() . . . 1540
9.237.6.12 create_completion_token() . . . 1540
9.237.6.13 delete_completion_token() . . . 1541
9.238 DDSAsyncWaitSetCompletionToken Class Reference . . . 1541
9.238.1 Detailed Description . . . 1541
9.238.2 AsyncWaitSetCompletionToken management . . . 1542
9.238.3 Member Function Documentation . . . 1542
9.238.3.1 wait() . . . 1542
9.239 DDSAsyncWaitSetListener Class Reference . . . 1543
9.239.1 Detailed Description . . . 1543
9.239.2 Member Function Documentation . . . 1543
9.239.2.1 on_thread_spawned() . . . 1543
9.239.2.2 on_thread_deleted() . . . 1544
9.239.2.3 on_wait_timeout() . . . 1544
9.240 DDSCondition Class Reference . . . 1544
9.240.1 Detailed Description . . . 1545
9.240.2 Member Function Documentation . . . 1545
9.240.2.1 get_trigger_value() . . . 1545
9.240.2.2 set_handler() . . . 1545
9.240.2.3 get_handler() . . . 1546
9.240.2.4 dispatch() . . . 1546
9.241 DDSConditionHandler Class Reference . . . 1546
9.241.1 Detailed Description . . . 1547
9.241.2 Member Function Documentation . . . 1547
9.241.2.1 on_condition_triggered() . . . 1547
9.242 DDSConditionSeq Struct Reference . . . 1547
9.242.1 Detailed Description . . . 1547
9.243 DDSContentFilter Class Reference . . . 1548
9.243.1 Detailed Description . . . 1548
9.243.2 Member Function Documentation . . . 1548
9.243.2.1 compile() . . . 1549
9.243.2.2 evaluate() . . . 1550
9.243.2.3 finalize() . . . 1550
9.244 DDSContentFilteredTopic Class Reference . . . 1551
9.244.1 Detailed Description . . . 1552
9.244.2 Member Function Documentation . . . 1552
9.244.2.1 narrow() . . . 1552

9.244.2.2 get_filter_expression() . . . 1553
9.244.2.3 get_expression_parameters() . . . 1553
9.244.2.4 set_expression_parameters() . . . 1554
9.244.2.5 set_expression() . . . 1554
9.244.2.6 append_to_expression_parameter() . . . 1554
9.244.2.7 remove_from_expression_parameter() . . . 1555
9.244.2.8 get_related_topic() . . . 1556
9.245 DDSDataReader Class Reference . . . 1556
9.245.1 Detailed Description . . . 1559
9.245.2 Member Function Documentation . . . 1561
9.245.2.1 create_readcondition() . . . 1561
9.245.2.2 create_readcondition_w_params() . . . 1561
9.245.2.3 create_querycondition() . . . 1562
9.245.2.4 create_querycondition_w_params() . . . 1562
9.245.2.5 delete_readcondition() . . . 1563
9.245.2.6 delete_contained_entities() . . . 1563
9.245.2.7 wait_for_historical_data() . . . 1564
9.245.2.8 acknowledge_sample() [1/2] . . . 1564
9.245.2.9 acknowledge_all() [1/2] . . . 1565
9.245.2.10 acknowledge_sample() [2/2] . . . 1565
9.245.2.11 acknowledge_all() [2/2] . . . 1566
9.245.2.12 get_matched_publications() . . . 1567
9.245.2.13 get_matched_publication_data() . . . 1568
9.245.2.14 is_matched_publication_alive() . . . 1568
9.245.2.15 get_matched_publication_participant_data() . . . 1569
9.245.2.16 get_topicdescription() . . . 1570
9.245.2.17 get_subscriber() . . . 1570
9.245.2.18 get_sample_rejected_status() . . . 1570
9.245.2.19 get_liveliness_changed_status() . . . 1571
9.245.2.20 get_requested_deadline_missed_status() . . . 1571
9.245.2.21 get_requested_incompatible_qos_status() . . . 1571
9.245.2.22 get_sample_lost_status() . . . 1572
9.245.2.23 get_subscription_matched_status() . . . 1572
9.245.2.24 get_datareader_cache_status() . . . 1573
9.245.2.25 get_datareader_protocol_status() . . . 1573
9.245.2.26 get_matched_publication_datareader_protocol_status() . . . 1573
9.245.2.27 set_qos() . . . 1574
9.245.2.28 get_qos() . . . 1575
9.245.2.29 set_qos_with_profile() . . . 1575

9.245.2.30 set_property() . . . 1576
9.245.2.31 set_listener() . . . 1577
9.245.2.32 get_listener() . . . 1577
9.245.2.33 create_topic_query() . . . 1578
9.245.2.34 delete_topic_query() . . . 1578
9.245.2.35 lookup_topic_query() . . . 1578
9.245.2.36 take_discovery_snapshot() [1/2] . . . 1579
9.245.2.37 take_discovery_snapshot() [2/2] . . . 1580
9.245.2.38 enable() . . . 1580
9.245.2.39 get_statuscondition() . . . 1581
9.245.2.40 get_status_changes() . . . 1582
9.245.2.41 get_instance_handle() . . . 1582
9.246 DDSDataReaderListener Class Reference . . . 1583
9.246.1 Detailed Description . . . 1583
9.246.2 Member Function Documentation . . . 1584
9.246.2.1 on_requested_deadline_missed() . . . 1584
9.246.2.2 on_liveliness_changed() . . . 1584
9.246.2.3 on_requested_incompatible_qos() . . . 1584
9.246.2.4 on_sample_rejected() . . . 1585
9.246.2.5 on_data_available() . . . 1585
9.246.2.6 on_sample_lost() . . . 1585
9.246.2.7 on_subscription_matched() . . . 1585
9.247 DDSDataReaderSeq Class Reference . . . 1585
9.247.1 Detailed Description . . . 1586
9.248 DDSDataReaderStatusConditionHandler Class Reference . . . 1586
9.248.1 Detailed Description . . . 1586
9.248.2 Constructor & Destructor Documentation . . . 1587
9.248.2.1 DDSDataReaderStatusConditionHandler() . . . 1587
9.248.2.2 ∼DDSDataReaderStatusConditionHandler() . . . 1588
9.248.3 Member Function Documentation . . . 1588
9.248.3.1 on_condition_triggered() . . . 1588
9.249 DDSDataTagQosPolicyHelper Class Reference . . . 1588
9.249.1 Detailed Description . . . 1588
9.250 DDSDataWriter Class Reference . . . 1589
9.250.1 Detailed Description . . . 1591
9.250.2 Member Function Documentation . . . 1592
9.250.2.1 get_liveliness_lost_status() . . . 1592
9.250.2.2 get_offered_deadline_missed_status() . . . 1592
9.250.2.3 get_offered_incompatible_qos_status() . . . 1593

9.250.2.4 get_publication_matched_status() . . . 1593
9.250.2.5 get_reliable_writer_cache_changed_status() . . . 1594
9.250.2.6 get_reliable_reader_activity_changed_status() . . . 1594
9.250.2.7 get_datawriter_cache_status() . . . 1595
9.250.2.8 get_datawriter_protocol_status() . . . 1595
9.250.2.9 get_matched_subscription_datawriter_protocol_status() . . . 1595
9.250.2.10 is_matched_subscription_active() . . . 1596
9.250.2.11 get_service_request_accepted_status() . . . 1597
9.250.2.12 get_matched_subscription_datawriter_protocol_status_by_locator() . . . 1597
9.250.2.13 assert_liveliness() . . . 1598
9.250.2.14 get_matched_subscription_locators() . . . 1598
9.250.2.15 get_matched_subscriptions() . . . 1599
9.250.2.16 get_matched_subscription_data() . . . 1600
9.250.2.17 get_matched_subscription_participant_data() . . . 1601
9.250.2.18 get_topic() . . . 1601
9.250.2.19 get_publisher() . . . 1602
9.250.2.20 wait_for_acknowledgments() . . . 1602
9.250.2.21 is_sample_app_acknowledged() . . . 1603
9.250.2.22 wait_for_asynchronous_publishing() . . . 1603
9.250.2.23 set_qos() . . . 1604
9.250.2.24 set_property() . . . 1605
9.250.2.25 set_qos_with_profile() . . . 1606
9.250.2.26 get_qos() . . . 1606
9.250.2.27 set_listener() . . . 1607
9.250.2.28 get_listener() . . . 1607
9.250.2.29 flush() . . . 1608
9.250.2.30 take_discovery_snapshot() [1/2] . . . 1608
9.250.2.31 take_discovery_snapshot() [2/2] . . . 1609
9.250.2.32 enable() . . . 1610
9.250.2.33 get_statuscondition() . . . 1611
9.250.2.34 get_status_changes() . . . 1611
9.250.2.35 get_instance_handle() . . . 1612
9.251 DDSDataWriterListener Class Reference . . . 1612
9.251.1 Detailed Description . . . 1613
9.251.2 Member Function Documentation . . . 1613
9.251.2.1 on_offered_deadline_missed() . . . 1614
9.251.2.2 on_liveliness_lost() . . . 1614
9.251.2.3 on_offered_incompatible_qos() . . . 1614
9.251.2.4 on_publication_matched() . . . 1615

9.251.2.5 on_reliable_writer_cache_changed() . . . 1615
9.251.2.6 on_reliable_reader_activity_changed() . . . 1616
9.251.2.7 on_sample_removed() . . . 1616
9.251.2.8 on_instance_replaced() . . . 1617
9.251.2.9 on_application_acknowledgment() . . . 1617
9.251.2.10 on_service_request_accepted() . . . 1618
9.252 DDSDomainEntity Class Reference . . . 1618
9.252.1 Detailed Description . . . 1619
9.253 DDSDomainParticipant Class Reference . . . 1619
9.253.1 Detailed Description . . . 1625
9.253.2 Member Function Documentation . . . 1626
9.253.2.1 get_default_datawriter_qos() . . . 1626
9.253.2.2 set_default_datawriter_qos() . . . 1627
9.253.2.3 set_default_datawriter_qos_with_profile() . . . 1628
9.253.2.4 get_default_datareader_qos() . . . 1628
9.253.2.5 set_default_datareader_qos() . . . 1629
9.253.2.6 set_default_datareader_qos_with_profile() . . . 1630
9.253.2.7 get_default_flowcontroller_property() . . . 1630
9.253.2.8 set_default_flowcontroller_property() . . . 1631
9.253.2.9 register_contentfilter() . . . 1632
9.253.2.10 lookup_contentfilter() . . . 1633
9.253.2.11 unregister_contentfilter() . . . 1633
9.253.2.12 get_default_library() . . . 1634
9.253.2.13 get_default_profile() . . . 1634
9.253.2.14 get_default_profile_library() . . . 1635
9.253.2.15 set_default_library() . . . 1635
9.253.2.16 set_default_profile() . . . 1636
9.253.2.17 get_default_topic_qos() . . . 1636
9.253.2.18 set_default_topic_qos() . . . 1637
9.253.2.19 set_default_topic_qos_with_profile() . . . 1638
9.253.2.20 get_default_publisher_qos() . . . 1639
9.253.2.21 set_default_publisher_qos() . . . 1640
9.253.2.22 set_default_publisher_qos_with_profile() . . . 1641
9.253.2.23 get_default_subscriber_qos() . . . 1642
9.253.2.24 set_default_subscriber_qos() . . . 1642
9.253.2.25 set_default_subscriber_qos_with_profile() . . . 1643
9.253.2.26 create_publisher() . . . 1644
9.253.2.27 create_publisher_with_profile() . . . 1645
9.253.2.28 delete_publisher() . . . 1646

9.253.2.29 create_subscriber() . . . 1647
9.253.2.30 create_subscriber_with_profile() . . . 1648
9.253.2.31 delete_subscriber() . . . 1649
9.253.2.32 get_publishers() . . . 1649
9.253.2.33 get_subscribers() . . . 1650
9.253.2.34 create_topic() . . . 1650
9.253.2.35 create_topic_with_profile() . . . 1652
9.253.2.36 delete_topic() . . . 1653
9.253.2.37 create_contentfilteredtopic() . . . 1654
9.253.2.38 create_contentfilteredtopic_with_filter() . . . 1654
9.253.2.39 delete_contentfilteredtopic() . . . 1655
9.253.2.40 create_multitopic() . . . 1656
9.253.2.41 delete_multitopic() . . . 1657
9.253.2.42 find_topic() . . . 1657
9.253.2.43 lookup_topicdescription() . . . 1658
9.253.2.44 create_flowcontroller() . . . 1659
9.253.2.45 delete_flowcontroller() . . . 1660
9.253.2.46 lookup_flowcontroller() . . . 1660
9.253.2.47 get_builtin_subscriber() . . . 1661
9.253.2.48 ignore_participant() . . . 1662
9.253.2.49 banish_ignored_participants() . . . 1662
9.253.2.50 ignore_topic() . . . 1663
9.253.2.51 ignore_publication() . . . 1664
9.253.2.52 ignore_subscription() . . . 1665
9.253.2.53 get_domain_id() . . . 1665
9.253.2.54 get_current_time() . . . 1666
9.253.2.55 register_durable_subscription() . . . 1666
9.253.2.56 delete_durable_subscription() . . . 1667
9.253.2.57 assert_liveliness() . . . 1667
9.253.2.58 resume_endpoint_discovery() . . . 1668
9.253.2.59 delete_contained_entities() . . . 1669
9.253.2.60 get_discovered_participants() . . . 1670
9.253.2.61 get_discovered_participants_from_subject_name() . . . 1670
9.253.2.62 get_discovered_participant_data() . . . 1671
9.253.2.63 get_discovered_participant_subject_name() . . . 1672
9.253.2.64 get_discovered_topics() . . . 1673
9.253.2.65 get_discovered_topic_data() . . . 1673
9.253.2.66 contains_entity() . . . 1674
9.253.2.67 get_typecode() . . . 1675

9.253.2.68 get_participant_protocol_status() . . . 1676
9.253.2.69 set_property() . . . 1676
9.253.2.70 set_qos() . . . 1677
9.253.2.71 set_qos_with_profile() . . . 1677
9.253.2.72 get_qos() . . . 1678
9.253.2.73 add_peer() . . . 1678
9.253.2.74 remove_peer() . . . 1680
9.253.2.75 get_dns_tracker_polling_period() . . . 1680
9.253.2.76 set_dns_tracker_polling_period() . . . 1681
9.253.2.77 set_listener() . . . 1682
9.253.2.78 get_listener() . . . 1682
9.253.2.79 get_implicit_publisher() . . . 1683
9.253.2.80 get_implicit_subscriber() . . . 1683
9.253.2.81 create_datawriter() . . . 1684
9.253.2.82 create_datawriter_with_profile() . . . 1685
9.253.2.83 delete_datawriter() . . . 1686
9.253.2.84 create_datareader() . . . 1687
9.253.2.85 create_datareader_with_profile() . . . 1688
9.253.2.86 delete_datareader() . . . 1689
9.253.2.87 lookup_publisher_by_name() . . . 1690
9.253.2.88 lookup_subscriber_by_name() . . . 1690
9.253.2.89 lookup_datawriter_by_name() . . . 1691
9.253.2.90 lookup_datareader_by_name() . . . 1692
9.253.2.91 take_discovery_snapshot() [1/2] . . . 1693
9.253.2.92 take_discovery_snapshot() [2/2] . . . 1693
9.254 DDSDomainParticipantFactory Class Reference . . . 1694
9.254.1 Detailed Description . . . 1697
9.254.2 Member Function Documentation . . . 1697
9.254.2.1 get_instance() . . . 1697
9.254.2.2 finalize_instance() . . . 1698
9.254.2.3 set_default_participant_qos() . . . 1698
9.254.2.4 set_default_participant_qos_with_profile() . . . 1699
9.254.2.5 get_default_participant_qos() . . . 1700
9.254.2.6 set_default_library() . . . 1701
9.254.2.7 get_default_library() . . . 1701
9.254.2.8 set_default_profile() . . . 1702
9.254.2.9 get_default_profile() . . . 1703
9.254.2.10 get_default_profile_library() . . . 1703
9.254.2.11 get_participant_factory_qos_from_profile() . . . 1704

9.254.2.12 get_participant_qos_from_profile() . . . 1704
9.254.2.13 get_publisher_qos_from_profile() . . . 1705
9.254.2.14 get_subscriber_qos_from_profile() . . . 1705
9.254.2.15 get_datawriter_qos_from_profile() . . . 1706
9.254.2.16 get_datawriter_qos_from_profile_w_topic_name() . . . 1706
9.254.2.17 get_datareader_qos_from_profile() . . . 1707
9.254.2.18 get_datareader_qos_from_profile_w_topic_name() . . . 1708
9.254.2.19 get_topic_qos_from_profile() . . . 1708
9.254.2.20 get_topic_qos_from_profile_w_topic_name() . . . 1709
9.254.2.21 get_qos_profile_libraries() . . . 1710
9.254.2.22 get_qos_profiles() . . . 1710
9.254.2.23 create_participant() . . . 1710
9.254.2.24 create_participant_with_profile() . . . 1712
9.254.2.25 delete_participant() . . . 1713
9.254.2.26 lookup_participant() . . . 1714
9.254.2.27 set_qos() . . . 1714
9.254.2.28 get_qos() . . . 1715
9.254.2.29 load_profiles() . . . 1715
9.254.2.30 reload_profiles() . . . 1716
9.254.2.31 unload_profiles() . . . 1716
9.254.2.32 unregister_thread() . . . 1716
9.254.2.33 get_typecode_from_config() . . . 1717
9.254.2.34 create_participant_from_config() . . . 1717
9.254.2.35 create_participant_from_config_w_params() . . . 1719
9.254.2.36 lookup_participant_by_name() . . . 1719
9.254.2.37 register_type_support() . . . 1720
9.254.2.38 get_participants() . . . 1721
9.254.2.39 set_thread_factory() . . . 1721
9.255 DDSDomainParticipantListener Class Reference . . . 1722
9.255.1 Detailed Description . . . 1723
9.255.2 Member Function Documentation . . . 1724
9.255.2.1 on_invalid_local_status_advance_notice() . . . 1724
9.256 DDSDynamicDataReader Class Reference . . . 1724
9.256.1 Detailed Description . . . 1724
9.257 DDSDynamicDataTypeSupport Class Reference . . . 1725
9.257.1 Detailed Description . . . 1725
9.257.2 Constructor & Destructor Documentation . . . 1726
9.257.2.1 DDSDynamicDataTypeSupport() . . . 1726
9.257.2.2 ∼DDSDynamicDataTypeSupport() . . . 1726

9.257.3 Member Function Documentation . . . 1727
9.257.3.1 is_valid() . . . 1727
9.257.3.2 register_type() . . . 1727
9.257.3.3 unregister_type() . . . 1728
9.257.3.4 get_type_name() . . . 1728
9.257.3.5 get_data_type() . . . 1728
9.257.3.6 create_data() . . . 1729
9.257.3.7 delete_data() . . . 1729
9.257.3.8 print_data() . . . 1729
9.257.3.9 copy_data() . . . 1730
9.258 DDSDynamicDataWriter Class Reference . . . 1730
9.258.1 Detailed Description . . . 1730
9.259 DDSEntity Class Reference . . . 1731
9.259.1 Detailed Description . . . 1731
9.259.2 Abstract operations . . . 1732
9.259.2.1 set_qos (abstract) . . . 1732
9.259.2.2 get_qos (abstract) . . . 1733
9.259.2.3 set_listener (abstract) . . . 1733
9.259.2.4 get_listener (abstract) . . . 1734
9.259.3 Member Function Documentation . . . 1734
9.259.3.1 enable() . . . 1734
9.259.3.2 get_statuscondition() . . . 1735
9.259.3.3 get_status_changes() . . . 1736
9.259.3.4 get_instance_handle() . . . 1736
9.260 DDSFlowController Class Reference . . . 1736
9.260.1 Detailed Description . . . 1737
9.260.2 Member Function Documentation . . . 1737
9.260.2.1 set_property() . . . 1737
9.260.2.2 get_property() . . . 1738
9.260.2.3 trigger_flow() . . . 1738
9.260.2.4 get_name() . . . 1739
9.260.2.5 get_participant() . . . 1739
9.261 DDSGuardCondition Class Reference . . . 1739
9.261.1 Detailed Description . . . 1740
9.261.2 Constructor & Destructor Documentation . . . 1740
9.261.2.1 DDSGuardCondition() . . . 1740
9.261.2.2 ~DDSGuardCondition() . . . 1740
9.261.3 Member Function Documentation . . . 1741
9.261.3.1 get_trigger_value() . . . 1741

9.261.3.2 set_trigger_value() . . . 1741
9.262 DDSKeyedOctetsDataReader Class Reference . . . 1741
9.262.1 Detailed Description . . . 1743
9.262.2 Member Function Documentation . . . 1744
9.262.2.1 read() . . . 1744
9.262.2.2 take() . . . 1744
9.262.2.3 read_w_condition() . . . 1744
9.262.2.4 take_w_condition() . . . 1745
9.262.2.5 read_next_sample() . . . 1745
9.262.2.6 take_next_sample() . . . 1745
9.262.2.7 read_instance() . . . 1746
9.262.2.8 take_instance() . . . 1746
9.262.2.9 read_instance_w_condition() . . . 1746
9.262.2.10 take_instance_w_condition() . . . 1747
9.262.2.11 read_next_instance() . . . 1747
9.262.2.12 take_next_instance() . . . 1747
9.262.2.13 read_next_instance_w_condition() . . . 1748
9.262.2.14 take_next_instance_w_condition() . . . 1748
9.262.2.15 return_loan() . . . 1748
9.262.2.16 get_key_value() [1/2] . . . 1749
9.262.2.17 get_key_value() [2/2] . . . 1749
9.262.2.18 lookup_instance() [1/2] . . . 1749
9.262.2.19 lookup_instance() [2/2] . . . 1750
9.262.2.20 narrow() . . . 1750
9.263 DDSKeyedOctetsDataWriter Class Reference . . . 1750
9.263.1 Detailed Description . . . 1753
9.263.2 Member Function Documentation . . . 1753
9.263.2.1 narrow() . . . 1753
9.263.2.2 register_instance() [1/2] . . . 1753
9.263.2.3 register_instance() [2/2] . . . 1754
9.263.2.4 register_instance_w_timestamp() [1/2] . . . 1754
9.263.2.5 register_instance_w_timestamp() [2/2] . . . 1754
9.263.2.6 unregister_instance() [1/2] . . . 1755
9.263.2.7 unregister_instance() [2/2] . . . 1755
9.263.2.8 unregister_instance_w_timestamp() [1/2] . . . 1755
9.263.2.9 unregister_instance_w_timestamp() [2/2] . . . 1756
9.263.2.10 create_data() . . . 1756
9.263.2.11 delete_data() . . . 1756
9.263.2.12 write() [1/3] . . . 1757

9.263.2.13 write() [2/3] . . . 1757
9.263.2.14 write() [3/3] . . . 1757
9.263.2.15 write_w_timestamp() [1/3] . . . 1758
9.263.2.16 write_w_timestamp() [2/3] . . . 1758
9.263.2.17 write_w_timestamp() [3/3] . . . 1759
9.263.2.18 write_w_params() [1/3] . . . 1759
9.263.2.19 write_w_params() [2/3] . . . 1760
9.263.2.20 write_w_params() [3/3] . . . 1760
9.263.2.21 dispose() [1/2] . . . 1761
9.263.2.22 dispose() [2/2] . . . 1761
9.263.2.23 dispose_w_timestamp() [1/2] . . . 1761
9.263.2.24 dispose_w_timestamp() [2/2] . . . 1762
9.263.2.25 get_key_value() [1/2] . . . 1762
9.263.2.26 get_key_value() [2/2] . . . 1762
9.263.2.27 lookup_instance() [1/2] . . . 1763
9.263.2.28 lookup_instance() [2/2] . . . 1763
9.264 DDSKeyedOctetsTypeSupport Class Reference . . . 1763
9.264.1 Detailed Description . . . 1764
9.264.2 Member Function Documentation . . . 1764
9.264.2.1 register_type() . . . 1764
9.264.2.2 unregister_type() . . . 1766
9.264.2.3 get_type_name() . . . 1767
9.264.2.4 print_data() . . . 1767
9.264.2.5 get_typecode() . . . 1768
9.264.2.6 serialize_data_to_cdr_buffer() . . . 1768
9.264.2.7 serialize_data_to_cdr_buffer_ex() . . . 1768
9.264.2.8 deserialize_data_from_cdr_buffer() . . . 1769
9.264.2.9 data_to_string() . . . 1769
9.265 DDSKeyedStringDataReader Class Reference . . . 1769
9.265.1 Detailed Description . . . 1771
9.265.2 Member Function Documentation . . . 1772
9.265.2.1 read() . . . 1772
9.265.2.2 take() . . . 1772
9.265.2.3 read_w_condition() . . . 1772
9.265.2.4 take_w_condition() . . . 1773
9.265.2.5 read_next_sample() . . . 1773
9.265.2.6 take_next_sample() . . . 1773
9.265.2.7 read_instance() . . . 1774
9.265.2.8 take_instance() . . . 1774

9.265.2.9 read_instance_w_condition() . . . 1774
9.265.2.10 take_instance_w_condition() . . . 1775
9.265.2.11 read_next_instance() . . . 1775
9.265.2.12 take_next_instance() . . . 1775
9.265.2.13 read_next_instance_w_condition() . . . 1776
9.265.2.14 take_next_instance_w_condition() . . . 1776
9.265.2.15 return_loan() . . . 1776
9.265.2.16 get_key_value() [1/2] . . . 1777
9.265.2.17 get_key_value() [2/2] . . . 1777
9.265.2.18 lookup_instance() [1/2] . . . 1777
9.265.2.19 lookup_instance() [2/2] . . . 1778
9.265.2.20 narrow() . . . 1778
9.266 DDSKeyedStringDataWriter Class Reference . . . 1778
9.266.1 Detailed Description . . . 1780
9.266.2 Member Function Documentation . . . 1780
9.266.2.1 narrow() . . . 1781
9.266.2.2 register_instance() [1/2] . . . 1781
9.266.2.3 register_instance() [2/2] . . . 1781
9.266.2.4 register_instance_w_timestamp() [1/2] . . . 1781
9.266.2.5 register_instance_w_timestamp() [2/2] . . . 1782
9.266.2.6 unregister_instance() [1/2] . . . 1782
9.266.2.7 unregister_instance() [2/2] . . . 1782
9.266.2.8 unregister_instance_w_timestamp() [1/2] . . . 1783
9.266.2.9 unregister_instance_w_timestamp() [2/2] . . . 1783
9.266.2.10 create_data() . . . 1783
9.266.2.11 delete_data() . . . 1784
9.266.2.12 write() [1/2] . . . 1784
9.266.2.13 write() [2/2] . . . 1784
9.266.2.14 write_w_timestamp() [1/2] . . . 1785
9.266.2.15 write_w_timestamp() [2/2] . . . 1785
9.266.2.16 write_w_params() [1/2] . . . 1785
9.266.2.17 write_w_params() [2/2] . . . 1786
9.266.2.18 dispose() [1/2] . . . 1786
9.266.2.19 dispose() [2/2] . . . 1786
9.266.2.20 dispose_w_timestamp() [1/2] . . . 1787
9.266.2.21 dispose_w_timestamp() [2/2] . . . 1787
9.266.2.22 get_key_value() [1/2] . . . 1787
9.266.2.23 get_key_value() [2/2] . . . 1788
9.266.2.24 lookup_instance() [1/2] . . . 1788

9.266.2.25 lookup_instance() [2/2] . . . 1788
9.267 DDSKeyedStringTypeSupport Class Reference . . . 1788
9.267.1 Detailed Description . . . 1789
9.267.2 Member Function Documentation . . . 1789
9.267.2.1 register_type() . . . 1789
9.267.2.2 unregister_type() . . . 1791
9.267.2.3 get_type_name() . . . 1792
9.267.2.4 print_data() . . . 1792
9.267.2.5 get_typecode() . . . 1793
9.267.2.6 serialize_data_to_cdr_buffer() . . . 1793
9.267.2.7 serialize_data_to_cdr_buffer_ex() . . . 1793
9.267.2.8 deserialize_data_from_cdr_buffer() . . . 1794
9.267.2.9 data_to_string() . . . 1794
9.268 DDSListener Class Reference . . . 1794
9.268.1 Detailed Description . . . 1795
9.268.2 Access to Plain Communication Status . . . 1795
9.268.3 Access to Read Communication Status . . . 1796
9.268.4 Operations Allowed in Listener Callbacks . . . 1796
9.268.5 Best Practices with Listeners . . . 1796
9.269 DDSMultiTopic Class Reference . . . 1797
9.269.1 Detailed Description . . . 1798
9.269.2 Member Function Documentation . . . 1799
9.269.2.1 narrow() . . . 1799
9.269.2.2 get_subscription_expression() . . . 1799
9.269.2.3 get_expression_parameters() . . . 1799
9.269.2.4 set_expression_parameters() . . . 1800
9.270 DDSOctetsDataReader Class Reference . . . 1800
9.270.1 Detailed Description . . . 1801
9.270.2 Member Function Documentation . . . 1802
9.270.2.1 read() . . . 1802
9.270.2.2 take() . . . 1802
9.270.2.3 read_w_condition() . . . 1802
9.270.2.4 take_w_condition() . . . 1803
9.270.2.5 read_next_sample() . . . 1803
9.270.2.6 take_next_sample() . . . 1803
9.270.2.7 return_loan() . . . 1804
9.270.2.8 narrow() . . . 1804
9.271 DDSOctetsDataWriter Class Reference . . . 1804
9.271.1 Detailed Description . . . 1805

9.271.2 Member Function Documentation . . . 1806
9.271.2.1 narrow() . . . 1806
9.271.2.2 create_data() . . . 1806
9.271.2.3 delete_data() . . . 1807
9.271.2.4 write() [1/3] . . . 1807
9.271.2.5 write() [2/3] . . . 1807
9.271.2.6 write() [3/3] . . . 1808
9.271.2.7 write_w_timestamp() [1/3] . . . 1808
9.271.2.8 write_w_timestamp() [2/3] . . . 1808
9.271.2.9 write_w_timestamp() [3/3] . . . 1809
9.271.2.10 write_w_params() [1/3] . . . 1809
9.271.2.11 write_w_params() [2/3] . . . 1810
9.271.2.12 write_w_params() [3/3] . . . 1810
9.272 DDSOctetsTypeSupport Class Reference . . . 1811
9.272.1 Detailed Description . . . 1811
9.272.2 Member Function Documentation . . . 1812
9.272.2.1 register_type() . . . 1812
9.272.2.2 unregister_type() . . . 1813
9.272.2.3 get_type_name() . . . 1813
9.272.2.4 print_data() . . . 1814
9.272.2.5 get_typecode() . . . 1814
9.272.2.6 serialize_data_to_cdr_buffer() . . . 1814
9.272.2.7 serialize_data_to_cdr_buffer_ex() . . . 1815
9.272.2.8 deserialize_data_from_cdr_buffer() . . . 1815
9.272.2.9 data_to_string() . . . 1815
9.273 DDSParticipantBuiltinTopicDataDataReader Class Reference . . . 1816
9.273.1 Detailed Description . . . 1816
9.274 DDSParticipantBuiltinTopicDataTypeSupport Class Reference . . . 1816
9.274.1 Detailed Description . . . 1816
9.275 DDSPropertyQosPolicyHelper Class Reference . . . 1817
9.275.1 Detailed Description . . . 1817
9.276 DDSPublicationBuiltinTopicDataDataReader Class Reference . . . 1817
9.276.1 Detailed Description . . . 1818
9.277 DDSPublicationBuiltinTopicDataTypeSupport Class Reference . . . 1818
9.277.1 Detailed Description . . . 1818
9.278 DDSPublisher Class Reference . . . 1818
9.278.1 Detailed Description . . . 1820
9.278.2 Member Function Documentation . . . 1821
9.278.2.1 get_default_datawriter_qos() . . . 1821

9.278.2.2 set_default_datawriter_qos() . . . 1822
9.278.2.3 set_default_datawriter_qos_with_profile() . . . 1823
9.278.2.4 set_default_library() . . . 1824
9.278.2.5 get_default_library() . . . 1824
9.278.2.6 set_default_profile() . . . 1825
9.278.2.7 get_default_profile() . . . 1825
9.278.2.8 get_default_profile_library() . . . 1826
9.278.2.9 create_datawriter() . . . 1826
9.278.2.10 create_datawriter_with_profile() . . . 1828
9.278.2.11 delete_datawriter() . . . 1829
9.278.2.12 lookup_datawriter() . . . 1830
9.278.2.13 get_all_datawriters() . . . 1830
9.278.2.14 suspend_publications() . . . 1831
9.278.2.15 resume_publications() . . . 1831
9.278.2.16 begin_coherent_changes() . . . 1832
9.278.2.17 end_coherent_changes() . . . 1833
9.278.2.18 get_participant() . . . 1833
9.278.2.19 delete_contained_entities() . . . 1834
9.278.2.20 copy_from_topic_qos() . . . 1834
9.278.2.21 wait_for_acknowledgments() . . . 1835
9.278.2.22 wait_for_asynchronous_publishing() . . . 1835
9.278.2.23 set_qos() . . . 1836
9.278.2.24 set_qos_with_profile() . . . 1837
9.278.2.25 get_qos() . . . 1837
9.278.2.26 set_listener() . . . 1838
9.278.2.27 get_listener() . . . 1838
9.278.2.28 lookup_datawriter_by_name() . . . 1839
9.279 DDSPublisherListener Class Reference . . . 1839
9.279.1 Detailed Description . . . 1840
9.280 DDSPublisherSeq Class Reference . . . 1840
9.280.1 Detailed Description . . . 1841
9.281 DDSQueryCondition Class Reference . . . 1841
9.281.1 Detailed Description . . . 1841
9.281.2 Member Function Documentation . . . 1842
9.281.2.1 get_query_expression() . . . 1842
9.281.2.2 get_query_parameters() . . . 1842
9.281.2.3 set_query_parameters() . . . 1842
9.282 DDSReadCondition Class Reference . . . 1842
9.282.1 Detailed Description . . . 1843

9.282.2 Member Function Documentation . . . 1843
9.282.2.1 get_sample_state_mask() . . . 1844
9.282.2.2 get_view_state_mask() . . . 1844
9.282.2.3 get_instance_state_mask() . . . 1844
9.282.2.4 get_stream_kind_mask() . . . 1844
9.282.2.5 get_datareader() . . . 1844
9.283 DDSServiceRequestDataReader Class Reference . . . 1845
9.283.1 Detailed Description . . . 1845
9.284 DDSServiceRequestTypeSupport Class Reference . . . 1845
9.284.1 Detailed Description . . . 1845
9.285 DDSStatusCondition Class Reference . . . 1846
9.285.1 Detailed Description . . . 1846
9.285.2 Member Function Documentation . . . 1846
9.285.2.1 get_enabled_statuses() . . . 1847
9.285.2.2 set_enabled_statuses() . . . 1847
9.285.2.3 get_entity() . . . 1847
9.286 DDSStringDataReader Class Reference . . . 1848
9.286.1 Detailed Description . . . 1849
9.286.2 Member Function Documentation . . . 1849
9.286.2.1 read() . . . 1849
9.286.2.2 take() . . . 1849
9.286.2.3 read_w_condition() . . . 1850
9.286.2.4 take_w_condition() . . . 1850
9.286.2.5 read_next_sample() . . . 1850
9.286.2.6 take_next_sample() . . . 1851
9.286.2.7 return_loan() . . . 1851
9.286.2.8 narrow() . . . 1851
9.287 DDSStringDataWriter Class Reference . . . 1852
9.287.1 Detailed Description . . . 1852
9.287.2 Member Function Documentation . . . 1853
9.287.2.1 narrow() . . . 1853
9.287.2.2 create_data() . . . 1853
9.287.2.3 delete_data() . . . 1854
9.287.2.4 write() . . . 1854
9.287.2.5 write_w_timestamp() . . . 1854
9.287.2.6 write_w_params() . . . 1855
9.288 DDSStringTypeSupport Class Reference . . . 1855
9.288.1 Detailed Description . . . 1856
9.288.2 Member Function Documentation . . . 1856

9.288.2.1 register_type() . . . 1856
9.288.2.2 unregister_type() . . . 1857
9.288.2.3 get_type_name() . . . 1858
9.288.2.4 print_data() . . . 1858
9.288.2.5 get_typecode() . . . 1858
9.288.2.6 serialize_data_to_cdr_buffer() . . . 1859
9.288.2.7 serialize_data_to_cdr_buffer_ex() . . . 1859
9.288.2.8 deserialize_data_from_cdr_buffer() . . . 1859
9.288.2.9 data_to_string() . . . 1860
9.289 DDSSubscriber Class Reference . . . 1860
9.289.1 Detailed Description . . . 1862
9.289.2 Member Function Documentation . . . 1863
9.289.2.1 get_default_datareader_qos() . . . 1863
9.289.2.2 set_default_datareader_qos() . . . 1863
9.289.2.3 set_default_datareader_qos_with_profile() . . . 1864
9.289.2.4 set_default_library() . . . 1865
9.289.2.5 get_default_library() . . . 1866
9.289.2.6 set_default_profile() . . . 1866
9.289.2.7 get_default_profile() . . . 1867
9.289.2.8 get_default_profile_library() . . . 1867
9.289.2.9 create_datareader() . . . 1868
9.289.2.10 create_datareader_with_profile() . . . 1869
9.289.2.11 delete_datareader() . . . 1871
9.289.2.12 delete_contained_entities() . . . 1871
9.289.2.13 lookup_datareader() . . . 1872
9.289.2.14 begin_access() . . . 1873
9.289.2.15 end_access() . . . 1874
9.289.2.16 get_datareaders() . . . 1874
9.289.2.17 get_all_datareaders() . . . 1875
9.289.2.18 notify_datareaders() . . . 1876
9.289.2.19 get_participant() . . . 1876
9.289.2.20 copy_from_topic_qos() . . . 1877
9.289.2.21 set_qos() . . . 1877
9.289.2.22 set_qos_with_profile() . . . 1878
9.289.2.23 get_qos() . . . 1879
9.289.2.24 set_listener() . . . 1879
9.289.2.25 get_listener() . . . 1880
9.289.2.26 lookup_datareader_by_name() . . . 1880
9.290 DDSSubscriberListener Class Reference . . . 1881

9.290.1 Detailed Description . . . 1881
9.290.2 Member Function Documentation . . . 1882
9.290.2.1 on_data_on_readers() . . . 1882
9.291 DDSSubscriberSeq Class Reference . . . 1882
9.291.1 Detailed Description . . . 1882
9.292 DDSSubscriptionBuiltinTopicDataDataReader Class Reference . . . 1882
9.292.1 Detailed Description . . . 1883
9.293 DDSSubscriptionBuiltinTopicDataTypeSupport Class Reference . . . 1883
9.293.1 Detailed Description . . . 1883
9.294 DDSThreadFactory Class Reference . . . 1883
9.294.1 Detailed Description . . . 1884
9.294.2 Member Function Documentation . . . 1884
9.294.2.1 create_thread() . . . 1884
9.294.2.2 delete_thread() . . . 1885
9.295 DDSTopic Class Reference . . . 1885
9.295.1 Detailed Description . . . 1886
9.295.2 Member Function Documentation . . . 1887
9.295.2.1 narrow() . . . 1887
9.295.2.2 get_inconsistent_topic_status() . . . 1887
9.295.2.3 set_qos() . . . 1887
9.295.2.4 set_qos_with_profile() . . . 1888
9.295.2.5 get_qos() . . . 1889
9.295.2.6 set_listener() . . . 1889
9.295.2.7 get_listener() . . . 1890
9.296 DDSTopicBuiltinTopicDataDataReader Class Reference . . . 1890
9.296.1 Detailed Description . . . 1891
9.297 DDSTopicBuiltinTopicDataTypeSupport Class Reference . . . 1891
9.297.1 Detailed Description . . . 1891
9.298 DDSTopicDescription Class Reference . . . 1892
9.298.1 Detailed Description . . . 1892
9.298.2 Member Function Documentation . . . 1892
9.298.2.1 get_type_name() . . . 1893
9.298.2.2 get_name() . . . 1893
9.298.2.3 get_participant() . . . 1894
9.299 DDSTopicListener Class Reference . . . 1894
9.299.1 Detailed Description . . . 1894
9.299.2 Member Function Documentation . . . 1895
9.299.2.1 on_inconsistent_topic() . . . 1895
9.300 DDSTopicQuery Class Reference . . . 1895

9.300.1 Detailed Description . . . 1895
9.300.2 Member Function Documentation . . . 1895
9.300.2.1 get_guid() . . . 1895
9.301 DDSTopicQueryHelper Class Reference . . . 1896
9.301.1 Detailed Description . . . 1896
9.301.2 Member Function Documentation . . . 1896
9.301.2.1 topic_query_data_from_service_request() . . . 1896
9.302 DDSTypeSupport Class Reference . . . 1897
9.302.1 Detailed Description . . . 1897
9.303 DDSWaitSet Class Reference . . . 1897
9.303.1 Detailed Description . . . 1898
9.303.2 Usage . . . 1898
9.303.3 Trigger State of a ::DDSStatusCondition . . . 1899
9.303.4 Trigger State of a ::DDSReadCondition . . . 1900
9.303.5 Trigger State of a ::DDSGuardCondition . . . 1900
9.303.6 Constructor & Destructor Documentation . . . 1900
9.303.6.1 ∼DDSWaitSet() . . . 1901
9.303.6.2 DDSWaitSet() [1/2] . . . 1901
9.303.6.3 DDSWaitSet() [2/2] . . . 1901
9.303.7 Member Function Documentation . . . 1901
9.303.7.1 wait() . . . 1902
9.303.7.2 attach_condition() . . . 1902
9.303.7.3 detach_condition() . . . 1903
9.303.7.4 get_conditions() . . . 1904
9.303.7.5 set_property() . . . 1904
9.303.7.6 get_property() . . . 1904
9.304 DDSWriterContentFilter Class Reference . . . 1905
9.304.1 Detailed Description . . . 1906
9.304.2 Member Function Documentation . . . 1906
9.304.2.1 writer_compile() . . . 1906
9.304.2.2 writer_evaluate() . . . 1907
9.304.2.3 writer_finalize() . . . 1908
9.304.2.4 writer_attach() . . . 1908
9.304.2.5 writer_detach() . . . 1909
9.304.2.6 writer_return_loan() . . . 1909
9.305 rti::flat::FinalAlignedArrayOffset< ElementOffset, N > Class Template Reference . . . 1909
9.305.1 Detailed Description . . . 1910
9.305.2 Member Function Documentation . . . 1911
9.305.2.1 get_element() . . . 1911

9.306 rti::flat::FinalArrayOffset< ElementOffset, N > Class Template Reference . . . 1911
9.306.1 Detailed Description . . . 1912
9.306.2 Member Function Documentation . . . 1912
9.306.2.1 get_element() . . . 1912
9.307 rti::flat::FinalOffset< T > Class Template Reference . . . 1913
9.307.1 Detailed Description . . . 1913
9.308 rti::flat::FinalSequenceBuilder< ElementOffset > Class Template Reference . . . 1913
9.308.1 Detailed Description . . . 1914
9.308.2 Member Function Documentation . . . 1914
9.308.2.1 add_next() . . . 1915
9.308.2.2 add_n() . . . 1915
9.308.2.3 finish() . . . 1915
9.309 rti::flat::flat_type_traits< T > Struct Template Reference . . . 1915
9.309.1 Detailed Description . . . 1915
9.310 Foo Struct Reference . . . 1916
9.310.1 Detailed Description . . . 1916
9.311 FooDataReader Class Reference . . . 1917
9.311.1 Detailed Description . . . 1918
9.311.2 Member Function Documentation . . . 1919
9.311.2.1 narrow() . . . 1919
9.311.2.2 read() . . . 1919
9.311.2.3 take() . . . 1920
9.311.2.4 read_w_condition() . . . 1925
9.311.2.5 take_w_condition() . . . 1926
9.311.2.6 read_next_sample() . . . 1927
9.311.2.7 take_next_sample() . . . 1928
9.311.2.8 read_instance() . . . 1929
9.311.2.9 take_instance() . . . 1931
9.311.2.10 read_instance_w_condition() . . . 1932
9.311.2.11 take_instance_w_condition() . . . 1934
9.311.2.12 read_next_instance() . . . 1935
9.311.2.13 take_next_instance() . . . 1936
9.311.2.14 read_next_instance_w_condition() . . . 1938
9.311.2.15 take_next_instance_w_condition() . . . 1939
9.311.2.16 return_loan() . . . 1940
9.311.2.17 get_key_value() . . . 1941
9.311.2.18 lookup_instance() . . . 1942
9.311.2.19 is_data_consistent() . . . 1943
9.312 FooDataWriter Class Reference . . . 1944

9.312.1 Detailed Description . . . 1945
9.312.2 Member Function Documentation . . . 1946
9.312.2.1 narrow() . . . 1946
9.312.2.2 register_instance() . . . 1946
9.312.2.3 register_instance_w_timestamp() . . . 1947
9.312.2.4 register_instance_w_params() . . . 1948
9.312.2.5 unregister_instance() . . . 1948
9.312.2.6 unregister_instance_w_timestamp() . . . 1950
9.312.2.7 unregister_instance_w_params() . . . 1951
9.312.2.8 write() . . . 1951
9.312.2.9 write_w_timestamp() . . . 1955
9.312.2.10 write_w_params() . . . 1956
9.312.2.11 dispose() . . . 1957
9.312.2.12 dispose_w_timestamp() . . . 1958
9.312.2.13 dispose_w_params() . . . 1959
9.312.2.14 get_key_value() . . . 1960
9.312.2.15 lookup_instance() . . . 1960
9.312.2.16 create_data() . . . 1962
9.312.2.17 delete_data() . . . 1962
9.312.2.18 get_loan() . . . 1963
9.312.2.19 discard_loan() . . . 1964
9.313 FooSeq Struct Reference . . . 1965
9.313.1 Detailed Description . . . 1966
9.313.2 Constructor & Destructor Documentation . . . 1966
9.313.2.1 ~FooSeq() . . . 1967
9.313.2.2 FooSeq() [1/2] . . . 1967
9.313.2.3 FooSeq() [2/2] . . . 1968
9.313.3 Member Function Documentation . . . 1968
9.313.3.1 operator=() . . . 1969
9.313.3.2 copy_no_alloc() . . . 1970
9.313.3.3 from_array() . . . 1971
9.313.3.4 to_array() . . . 1972
9.313.3.5 operator[]() [1/2] . . . 1972
9.313.3.6 operator[]() [2/2] . . . 1973
9.313.3.7 length() [1/2] . . . 1973
9.313.3.8 length() [2/2] . . . 1973
9.313.3.9 ensure_length() . . . 1974
9.313.3.10 maximum() [1/2] . . . 1975
9.313.3.11 maximum() [2/2] . . . 1975

9.313.3.12 loan_contiguous() . . . 1976
9.313.3.13 loan_discontiguous() . . . 1977
9.313.3.14 unloan() . . . 1978
9.313.3.15 get_contiguous_buffer() . . . 1978
9.313.3.16 get_discontiguous_buffer() . . . 1979
9.313.3.17 has_ownership() . . . 1979
9.314 FooTypeSupport Class Reference . . . 1979
9.314.1 Detailed Description . . . 1981
9.314.2 Member Function Documentation . . . 1981
9.314.2.1 register_type() . . . 1981
9.314.2.2 unregister_type() . . . 1982
9.314.2.3 create_data() . . . 1983
9.314.2.4 create_data_ex() . . . 1983
9.314.2.5 copy_data() . . . 1984
9.314.2.6 delete_data() . . . 1984
9.314.2.7 delete_data_ex() . . . 1985
9.314.2.8 initialize_data() . . . 1986
9.314.2.9 initialize_data_ex() . . . 1986
9.314.2.10 finalize_data() . . . 1987
9.314.2.11 finalize_data_ex() . . . 1988
9.314.2.12 get_type_name() . . . 1988
9.314.2.13 print_data() . . . 1989
9.314.2.14 serialize_data_to_cdr_buffer() . . . 1989
9.314.2.15 serialize_data_to_cdr_buffer_ex() . . . 1990
9.314.2.16 deserialize_data_from_cdr_buffer() . . . 1990
9.314.2.17 data_to_string() . . . 1991
9.314.2.18 get_typecode() . . . 1992
9.315 connext::IllegalOperationException Class Reference . . . 1992
9.315.1 Detailed Description . . . 1992
9.316 connext::ImmutablePolicyException Class Reference . . . 1992
9.316.1 Detailed Description . . . 1993
9.317 connext::InconsistentPolicyException Class Reference . . . 1993
9.317.1 Detailed Description . . . 1993
9.318 connext::IsInvalidSamplePredicate< T > Class Template Reference . . . 1993
9.318.1 Detailed Description . . . 1993
9.319 connext::IsReplyRelatedPredicate< T > Class Template Reference . . . 1993
9.319.1 Detailed Description . . . 1994
9.319.2 Constructor & Destructor Documentation . . . 1994
9.319.2.1 IsReplyRelatedPredicate() . . . 1994

9.319.3 Member Function Documentation . . . 1994
9.319.3.1 operator()() . . . 1994
9.320 connext::IsValidSamplePredicate< T > Class Template Reference . . . 1995
9.320.1 Detailed Description . . . 1995
9.321 connext::LoanedSamples< T > Class Template Reference . . . 1995
9.321.1 Detailed Description . . . 1996
9.321.2 Member Typedef Documentation . . . 1997
9.321.2.1 iterator . . . 1998
9.321.2.2 const_iterator . . . 1998
9.321.3 Constructor & Destructor Documentation . . . 1998
9.321.3.1 LoanedSamples() [1/2] . . . 1998
9.321.3.2 LoanedSamples() [2/2] . . . 1998
9.321.3.3 ~LoanedSamples() . . . 1999
9.321.4 Member Function Documentation . . . 1999
9.321.4.1 move_construct_from_loans() . . . 1999
9.321.4.2 release() . . . 2000
9.321.4.3 operator[]() [1/2] . . . 2001
9.321.4.4 operator[]() [2/2] . . . 2001
9.321.4.5 length() . . . 2002
9.321.4.6 return_loan() . . . 2002
9.321.4.7 operator LoanMemento() . . . 2003
9.321.4.8 begin() [1/2] . . . 2003
9.321.4.9 end() [1/2] . . . 2003
9.321.4.10 begin() [2/2] . . . 2004
9.321.4.11 end() [2/2] . . . 2004
9.322 connext::LoanedSamples< T >::LoanMemento Struct Reference . . . 2004
9.322.1 Detailed Description . . . 2004
9.323 connext::LogicException Class Reference . . . 2005
9.323.1 Detailed Description . . . 2005
9.324 connext::MessagingLibraryVersion Class Reference . . . 2005
9.324.1 Detailed Description . . . 2006
9.324.2 Member Function Documentation . . . 2006
9.324.2.1 major_version() . . . 2006
9.324.2.2 minor_version() . . . 2006
9.324.2.3 release_version() . . . 2006
9.324.2.4 build_version() . . . 2006
9.325 connext::MessagingVersion Class Reference . . . 2006
9.325.1 Detailed Description . . . 2007
9.325.2 Member Function Documentation . . . 2007

9.325.2.1 get_api_version() . . . 2007
9.325.2.2 get_api_version_string() . . . 2007
9.325.2.3 get_api_build_version() . . . 2007
9.326 rti::flat::MutableArrayBuilder< ElementBuilder, N > Class Template Reference . . . 2008
9.326.1 Detailed Description . . . 2008
9.326.2 Member Typedef Documentation . . . 2009
9.326.2.1 Offset . . . 2009
9.326.3 Member Function Documentation . . . 2009
9.326.3.1 build_next() . . . 2009
9.326.3.2 finish() . . . 2010
9.327 rti::flat::MutableArrayOffset< ElementOffset, N > Class Template Reference . . . 2010
9.327.1 Detailed Description . . . 2010
9.327.2 Member Function Documentation . . . 2011
9.327.2.1 get_element() . . . 2011
9.328 rti::flat::MutableOffset Class Reference . . . 2011
9.328.1 Detailed Description . . . 2012
9.329 rti::flat::MutableSequenceBuilder< ElementBuilder > Class Template Reference . . . 2012
9.329.1 Detailed Description . . . 2012
9.329.2 Member Typedef Documentation . . . 2013
9.329.2.1 Offset . . . 2013
9.329.3 Member Function Documentation . . . 2013
9.329.3.1 build_next() . . . 2013
9.329.3.2 finish() . . . 2014
9.330 MyFlatFinalOffset Class Reference . . . 2014
9.330.1 Detailed Description . . . 2015
9.330.2 Member Typedef Documentation . . . 2015
9.330.2.1 ConstOffset . . . 2015
9.330.3 Constructor & Destructor Documentation . . . 2016
9.330.3.1 MyFlatFinalOffset() . . . 2016
9.330.4 Member Function Documentation . . . 2016
9.330.4.1 my_primitive() [1/2] . . . 2016
9.330.4.2 my_complex() [1/2] . . . 2016
9.330.4.3 my_primitive_array() [1/2] . . . 2016
9.330.4.4 my_complex_array() [1/2] . . . 2017
9.330.4.5 my_primitive() [2/2] . . . 2017
9.330.4.6 my_complex() [2/2] . . . 2017
9.330.4.7 my_primitive_array() [2/2] . . . 2017
9.330.4.8 my_complex_array() [2/2] . . . 2017
9.331 MyFlatMutableBuilder Class Reference . . . 2018

9.331.1 Detailed Description . . . 2019
9.331.1.1 Adding fixed-size members . . . 2019
9.331.1.2 Building variable-size members . . . 2019
9.331.1.3 Choosing which members are included . . . 2020
9.331.2 Member Typedef Documentation . . . 2020
9.331.2.1 Offset . . . 2020
9.331.3 Constructor & Destructor Documentation . . . 2020
9.331.3.1 MyFlatMutableBuilder() [1/2] . . . 2020
9.331.3.2 MyFlatMutableBuilder() [2/2] . . . 2021
9.331.4 Member Function Documentation . . . 2021
9.331.4.1 finish() . . . 2021
9.331.4.2 finish_sample() . . . 2022
9.331.4.3 add_my_primitive() . . . 2022
9.331.4.4 add_my_optional_primitive() . . . 2022
9.331.4.5 add_my_primitive_array() . . . 2023
9.331.4.6 build_my_primitive_seq() . . . 2023
9.331.4.7 add_my_final() . . . 2023
9.331.4.8 add_my_final_array() . . . 2023
9.331.4.9 build_my_final_seq() . . . 2024
9.331.4.10 build_my_mutable() . . . 2024
9.331.4.11 build_my_mutable_array() . . . 2024
9.331.4.12 build_my_mutable_seq() . . . 2024
9.331.4.13 build_my_string() . . . 2024
9.331.4.14 build_my_string_seq() . . . 2025
9.332 MyFlatMutableOffset Class Reference . . . 2025
9.332.1 Detailed Description . . . 2026
9.332.2 Member Typedef Documentation . . . 2026
9.332.2.1 ConstOffset . . . 2027
9.332.3 Constructor & Destructor Documentation . . . 2027
9.332.3.1 MyFlatMutableOffset() . . . 2027
9.332.4 Member Function Documentation . . . 2027
9.332.4.1 my_primitive() [1/2] . . . 2027
9.332.4.2 my_optional_primitive() . . . 2027
9.332.4.3 my_primitive_array() [1/2] . . . 2028
9.332.4.4 my_primitive_seq() [1/2] . . . 2028
9.332.4.5 my_final() [1/2] . . . 2028
9.332.4.6 my_final_array() [1/2] . . . 2028
9.332.4.7 my_final_seq() [1/2] . . . 2028
9.332.4.8 my_mutable() [1/2] . . . 2029

9.332.4.9 my_mutable_array() [1/2] . . . 2029
9.332.4.10 my_mutable_seq() [1/2] . . . 2029
9.332.4.11 my_string() [1/2] . . . 2029
9.332.4.12 my_string_seq() [1/2] . . . 2029
9.332.4.13 my_primitive() [2/2] . . . 2030
9.332.4.14 my_primitive_array() [2/2] . . . 2030
9.332.4.15 my_primitive_seq() [2/2] . . . 2030
9.332.4.16 my_final() [2/2] . . . 2030
9.332.4.17 my_final_array() [2/2] . . . 2030
9.332.4.18 my_final_seq() [2/2] . . . 2031
9.332.4.19 my_mutable() [2/2] . . . 2031
9.332.4.20 my_mutable_array() [2/2] . . . 2031
9.332.4.21 my_mutable_seq() [2/2] . . . 2031
9.332.4.22 my_string() [2/2] . . . 2031
9.332.4.23 my_string_seq() [2/2] . . . 2031
9.333 MyFlatUnionBuilder Class Reference . . . 2032
9.333.1 Detailed Description . . . 2032
9.333.2 Member Typedef Documentation . . . 2033
9.333.2.1 Offset . . . 2033
9.333.3 Constructor & Destructor Documentation . . . 2033
9.333.3.1 MyFlatUnionBuilder() . . . 2033
9.333.4 Member Function Documentation . . . 2033
9.333.4.1 finish() . . . 2033
9.333.4.2 finish_sample() . . . 2034
9.333.4.3 add_my_primitive() . . . 2034
9.333.4.4 build_my_mutable() . . . 2034
9.333.4.5 add_my_final() . . . 2034
9.334 MyFlatUnionOffset Class Reference . . . 2035
9.334.1 Detailed Description . . . 2036
9.334.2 Member Typedef Documentation . . . 2036
9.334.2.1 ConstOffset . . . 2036
9.334.3 Constructor & Destructor Documentation . . . 2036
9.334.3.1 MyFlatUnionOffset() . . . 2036
9.334.4 Member Function Documentation . . . 2036
9.334.4.1 _d() . . . 2037
9.334.4.2 my_primitive() [1/2] . . . 2037
9.334.4.3 my_mutable() [1/2] . . . 2037
9.334.4.4 my_final() [1/2] . . . 2038
9.334.4.5 my_primitive() [2/2] . . . 2038

9.334.4.6 my_mutable() [2/2] . . . 2038
9.334.4.7 my_final() [2/2] . . . 2038
9.335 NDDS_Config_LibraryVersion_t Struct Reference . . . 2039
9.335.1 Detailed Description . . . 2039
9.335.2 Member Data Documentation . . . 2039
9.335.2.1 major . . . 2039
9.335.2.2 minor . . . 2039
9.335.2.3 release . . . 2040
9.335.2.4 build . . . 2040
9.336 NDDS_Config_LogMessage Struct Reference . . . 2040
9.336.1 Detailed Description . . . 2040
9.336.2 Member Data Documentation . . . 2040
9.336.2.1 text . . . 2041
9.336.2.2 level . . . 2041
9.336.2.3 is_security_message . . . 2041
9.336.2.4 is_license_message . . . 2041
9.336.2.5 message_id . . . 2042
9.336.2.6 timestamp . . . 2042
9.336.2.7 facility . . . 2042
9.337 NDDS_Transport_Address_t Struct Reference . . . 2042
9.337.1 Detailed Description . . . 2042
9.337.2 Member Data Documentation . . . 2043
9.337.2.1 network_ordered_value . . . 2043
9.338 NDDS_Transport_Interface_t Struct Reference . . . 2043
9.338.1 Detailed Description . . . 2043
9.338.2 Member Data Documentation . . . 2043
9.338.2.1 transport_classid . . . 2043
9.338.2.2 address . . . 2044
9.338.2.3 status . . . 2044
9.338.2.4 rank . . . 2044
9.339 NDDS_Transport_Property_t Struct Reference . . . 2044
9.339.1 Detailed Description . . . 2045
9.339.2 Member Data Documentation . . . 2045
9.339.2.1 classid . . . 2046
9.339.2.2 address_bit_count . . . 2046
9.339.2.3 properties_bitmap . . . 2047
9.339.2.4 gather_send_buffer_count_max . . . 2047
9.339.2.5 message_size_max . . . 2047
9.339.2.6 allow_interfaces_list . . . 2048

9.339.2.7 allow_interfaces_list_length . . . 2048
9.339.2.8 deny_interfaces_list . . . 2049
9.339.2.9 deny_interfaces_list_length . . . 2049
9.339.2.10 allow_multicast_interfaces_list . . . 2050
9.339.2.11 allow_multicast_interfaces_list_length . . . 2050
9.339.2.12 deny_multicast_interfaces_list . . . 2050
9.339.2.13 deny_multicast_interfaces_list_length . . . 2051
9.339.2.14 transport_uuid . . . 2051
9.339.2.15 thread_name_prefix . . . 2051
9.339.2.16 max_interface_count . . . 2052
9.340 NDDS_Transport_Shmem_Property_t Struct Reference . . . 2052
9.340.1 Detailed Description . . . 2053
9.340.2 Member Data Documentation . . . 2053
9.340.2.1 parent . . . 2053
9.340.2.2 received_message_count_max . . . 2053
9.340.2.3 receive_buffer_size . . . 2054
9.340.2.4 enable_udp_debugging . . . 2054
9.340.2.5 udp_debugging_address . . . 2055
9.340.2.6 udp_debugging_port . . . 2055
9.341 NDDS_Transport_UDP_WAN_CommPortsMappingInfo Struct Reference . . . 2055
9.341.1 Detailed Description . . . 2055
9.341.2 Member Data Documentation . . . 2055
9.341.2.1 rtps_port . . . 2056
9.341.2.2 host_port . . . 2056
9.341.2.3 public_port . . . 2056
9.342 NDDS_Transport_UDPv4_Property_t Struct Reference . . . 2056
9.342.1 Detailed Description . . . 2057
9.342.2 Member Data Documentation . . . 2058
9.342.2.1 parent . . . 2058
9.342.2.2 send_socket_buffer_size . . . 2058
9.342.2.3 recv_socket_buffer_size . . . 2059
9.342.2.4 unicast_enabled . . . 2059
9.342.2.5 multicast_enabled . . . 2059
9.342.2.6 multicast_ttl . . . 2060
9.342.2.7 multicast_loopback_disabled . . . 2060
9.342.2.8 ignore_loopback_interface . . . 2060
9.342.2.9 ignore_nonup_interfaces . . . 2061
9.342.2.10 ignore_nonrunning_interfaces . . . 2061
9.342.2.11 no_zero_copy . . . 2062

9.342.2.12 send_blocking . . . 2062
9.342.2.13 use_checksum . . . 2062
9.342.2.14 transport_priority_mask . . . 2063
9.342.2.15 transport_priority_mapping_low . . . 2063
9.342.2.16 transport_priority_mapping_high . . . 2063
9.342.2.17 send_ping . . . 2064
9.342.2.18 force_interface_poll_detection . . . 2064
9.342.2.19 interface_poll_period . . . 2064
9.342.2.20 reuse_multicast_receive_resource . . . 2064
9.342.2.21 protocol_overhead_max . . . 2065
9.342.2.22 disable_interface_tracking . . . 2065
9.342.2.23 join_multicast_group_timeout . . . 2065
9.342.2.24 public_address . . . 2066
9.342.2.25 port_offset . . . 2066
9.343 NDDS_Transport_UDPv4_WAN_Property_t Struct Reference . . . 2067
9.343.1 Detailed Description . . . 2068
9.343.2 Member Data Documentation . . . 2068
9.343.2.1 parent . . . 2068
9.343.2.2 send_socket_buffer_size . . . 2068
9.343.2.3 recv_socket_buffer_size . . . 2069
9.343.2.4 ignore_loopback_interface . . . 2069
9.343.2.5 ignore_nonup_interfaces . . . 2070
9.343.2.6 ignore_nonrunning_interfaces . . . 2070
9.343.2.7 no_zero_copy . . . 2071
9.343.2.8 send_blocking . . . 2071
9.343.2.9 use_checksum . . . 2071
9.343.2.10 transport_priority_mask . . . 2072
9.343.2.11 transport_priority_mapping_low . . . 2072
9.343.2.12 transport_priority_mapping_high . . . 2072
9.343.2.13 send_ping . . . 2073
9.343.2.14 force_interface_poll_detection . . . 2073
9.343.2.15 interface_poll_period . . . 2073
9.343.2.16 protocol_overhead_max . . . 2073
9.343.2.17 disable_interface_tracking . . . 2074
9.343.2.18 public_address . . . 2074
9.343.2.19 comm_ports_list . . . 2075
9.343.2.20 comm_ports_list_length . . . 2076
9.343.2.21 port_offset . . . 2076
9.343.2.22 binding_ping_period . . . 2076

9.344 NDDS_Transport_UDPv6_Property_t Struct Reference . . . 2077
9.344.1 Detailed Description . . . 2078
9.344.2 Member Data Documentation . . . 2078
9.344.2.1 parent . . . 2078
9.344.2.2 send_socket_buffer_size . . . 2079
9.344.2.3 recv_socket_buffer_size . . . 2079
9.344.2.4 unicast_enabled . . . 2080
9.344.2.5 multicast_enabled . . . 2080
9.344.2.6 multicast_ttl . . . 2080
9.344.2.7 multicast_loopback_disabled . . . 2080
9.344.2.8 ignore_loopback_interface . . . 2081
9.344.2.9 ignore_nonrunning_interfaces . . . 2081
9.344.2.10 no_zero_copy . . . 2082
9.344.2.11 send_blocking . . . 2082
9.344.2.12 enable_v4mapped . . . 2082
9.344.2.13 transport_priority_mask . . . 2083
9.344.2.14 transport_priority_mapping_low . . . 2083
9.344.2.15 transport_priority_mapping_high . . . 2083
9.344.2.16 send_ping . . . 2084
9.344.2.17 force_interface_poll_detection . . . 2084
9.344.2.18 interface_poll_period . . . 2084
9.344.2.19 reuse_multicast_receive_resource . . . 2084
9.344.2.20 protocol_overhead_max . . . 2085
9.344.2.21 disable_interface_tracking . . . 2085
9.344.2.22 join_multicast_group_timeout . . . 2085
9.344.2.23 public_address . . . 2086
9.344.2.24 port_offset . . . 2086
9.345 NDDS_Transport_UUID Struct Reference . . . 2087
9.345.1 Detailed Description . . . 2087
9.346 NDDS_Utility_NetworkCaptureParams_t Struct Reference . . . 2087
9.346.1 Detailed Description . . . 2087
9.346.2 Member Data Documentation . . . 2087
9.346.2.1 transports . . . 2088
9.346.2.2 dropped_content . . . 2088
9.346.2.3 traffic . . . 2088
9.346.2.4 parse_encrypted_content . . . 2088
9.346.2.5 checkpoint_thread_settings . . . 2089
9.346.2.6 frame_queue_size . . . 2089
9.347 NDDSConfigActivityContext Class Reference . . . 2089

9.347.1 Detailed Description . . . 2090
9.348 NDDSConfigCompliance Class Reference . . . 2090
9.348.1 Detailed Description . . . 2090
9.348.2 Member Function Documentation . . . 2090
9.348.2.1 get_xtypes_mask() . . . 2090
9.348.2.2 set_xtypes_mask() . . . 2091
9.348.2.3 load_compliance_masks() . . . 2091
9.348.2.4 get_rpc_mask() . . . 2092
9.348.2.5 set_rpc_mask() . . . 2092
9.349 NDDSConfigLogger Class Reference . . . 2093
9.349.1 Detailed Description . . . 2094
9.349.2 Member Function Documentation . . . 2094
9.349.2.1 get_instance() . . . 2094
9.349.2.2 finalize_instance() . . . 2095
9.349.2.3 get_verbosity() . . . 2095
9.349.2.4 get_verbosity_by_category() . . . 2095
9.349.2.5 set_verbosity() . . . 2095
9.349.2.6 set_verbosity_by_category() . . . 2096
9.349.2.7 get_output_file() . . . 2096
9.349.2.8 set_output_file() . . . 2096
9.349.2.9 set_output_file_set() . . . 2096
9.349.2.10 get_output_device() . . . 2097
9.349.2.11 set_output_device() . . . 2097
9.349.2.12 get_print_format() . . . 2097
9.349.2.13 set_print_format() . . . 2098
9.349.2.14 get_print_format_by_log_level() . . . 2098
9.349.2.15 set_print_format_by_log_level() . . . 2098
9.349.2.16 emergency() . . . 2098
9.349.2.17 alert() . . . 2099
9.349.2.18 critical() . . . 2099
9.349.2.19 error() . . . 2099
9.349.2.20 warning() . . . 2099
9.349.2.21 notice() . . . 2099
9.349.2.22 informational() . . . 2100
9.349.2.23 debug() . . . 2100
9.350 NDDSConfigLoggerDevice Class Reference . . . 2100
9.350.1 Detailed Description . . . 2100
9.350.2 Member Function Documentation . . . 2101
9.350.2.1 write() . . . 2101

9.350.2.2 close() . . . 2101
9.351 NDDSConfigVersion Class Reference . . . 2101
9.351.1 Detailed Description . . . 2102
9.351.2 Member Function Documentation . . . 2102
9.351.2.1 get_instance() . . . 2102
9.351.2.2 get_product_version() . . . 2102
9.351.2.3 get_cpp_api_version() . . . 2102
9.351.2.4 get_c_api_version() . . . 2102
9.351.2.5 get_core_version() . . . 2103
9.351.2.6 to_string() . . . 2103
9.352 NDDSLM Class Reference . . . 2103
9.352.1 Detailed Description . . . 2103
9.352.2 Member Function Documentation . . . 2103
9.352.2.1 unload_activation_library() . . . 2103
9.353 NDDSTransportSupport Class Reference . . . 2104
9.353.1 Detailed Description . . . 2104
9.353.2 Member Function Documentation . . . 2104
9.353.2.1 register_transport() . . . 2105
9.353.2.2 lookup_transport() . . . 2106
9.353.2.3 add_send_route() . . . 2107
9.353.2.4 add_receive_route() . . . 2107
9.353.2.5 get_builtin_transport_property() . . . 2108
9.353.2.6 set_builtin_transport_property() . . . 2109
9.353.2.7 get_transport_plugin() . . . 2110
9.354 NDDSUtility Class Reference . . . 2110
9.354.1 Detailed Description . . . 2111
9.354.2 Member Function Documentation . . . 2111
9.354.2.1 sleep() . . . 2111
9.354.2.2 spin() . . . 2111
9.354.2.3 get_spin_per_microsecond() . . . 2112
9.354.2.4 enable_heap_monitoring() . . . 2112
9.354.2.5 disable_heap_monitoring() . . . 2112
9.354.2.6 pause_heap_monitoring() . . . 2113
9.354.2.7 resume_heap_monitoring() . . . 2113
9.354.2.8 take_heap_snapshot() . . . 2113
9.355 NDDSUtilityHeapMonitoring Class Reference . . . 2113
9.355.1 Detailed Description . . . 2114
9.355.2 Member Function Documentation . . . 2114
9.355.2.1 enable() [1/2] . . . 2114

9.355.2.2 enable() [2/2] . . . 2114
9.355.2.3 disable() . . . 2115
9.355.2.4 pause() . . . 2115
9.355.2.5 resume() . . . 2115
9.355.2.6 take_heap_snapshot() . . . 2116
9.356 NDDSUtilityNetworkCapture Class Reference . . . 2117
9.356.1 Detailed Description . . . 2118
9.356.2 Member Function Documentation . . . 2118
9.356.2.1 enable() . . . 2118
9.356.2.2 disable() . . . 2119
9.356.2.3 set_default_params() . . . 2119
9.356.2.4 start() [1/4] . . . 2120
9.356.2.5 start() [2/4] . . . 2121
9.356.2.6 start() [3/4] . . . 2121
9.356.2.7 start() [4/4] . . . 2122
9.356.2.8 stop() [1/2] . . . 2123
9.356.2.9 stop() [2/2] . . . 2123
9.356.2.10 pause() [1/2] . . . 2124
9.356.2.11 pause() [2/2] . . . 2124
9.356.2.12 resume() [1/2] . . . 2125
9.356.2.13 resume() [2/2] . . . 2125
9.357 connext::NotEnabledException Class Reference . . . 2126
9.357.1 Detailed Description . . . 2126
9.358 rti::flat::OffsetBase Class Reference . . . 2126
9.358.1 Detailed Description . . . 2128
9.358.2 Member Function Documentation . . . 2128
9.358.2.1 is_null() . . . 2128
9.358.2.2 is_cpp_compatible() . . . 2128
9.358.2.3 get_buffer() . . . 2129
9.358.2.4 get_buffer_size() . . . 2129
9.358.3 Friends And Related Function Documentation . . . 2129
9.358.3.1 operator< . . . 2129
9.358.3.2 operator> . . . 2130
9.358.3.3 operator<= . . . 2130
9.358.3.4 operator>= . . . 2130
9.358.3.5 operator== . . . 2131
9.358.3.6 operator"!= . . . 2131
9.359 connext::OutOfResourcesException Class Reference . . . 2131
9.359.1 Detailed Description . . . 2131

9.360 connext::PreconditionNotMetException Class Reference . . . 2132
9.360.1 Detailed Description . . . 2132
9.361 rti::flat::PrimitiveArrayOffset< T, N > Class Template Reference . . . 2132
9.361.1 Detailed Description . . . 2132
9.361.2 Member Function Documentation . . . 2133
9.361.2.1 element_count() . . . 2133
9.362 rti::flat::PrimitiveConstOffset< T > Struct Template Reference . . . 2133
9.362.1 Detailed Description . . . 2134
9.362.2 Member Function Documentation . . . 2134
9.362.2.1 get() . . . 2134
9.363 rti::flat::PrimitiveOffset< T > Struct Template Reference . . . 2134
9.363.1 Detailed Description . . . 2134
9.363.2 Member Function Documentation . . . 2135
9.363.2.1 set() . . . 2135
9.364 rti::flat::PrimitiveSequenceBuilder< T > Class Template Reference . . . 2135
9.364.1 Detailed Description . . . 2136
9.364.2 Member Function Documentation . . . 2136
9.364.2.1 add_next() . . . 2136
9.364.2.2 add_n() [1/3] . . . 2137
9.364.2.3 add_n() [2/3] . . . 2137
9.364.2.4 add_n() [3/3] . . . 2137
9.364.2.5 finish() . . . 2138
9.365 rti::flat::PrimitiveSequenceOffset< T > Class Template Reference . . . 2138
9.365.1 Detailed Description . . . 2139
9.365.2 Member Function Documentation . . . 2139
9.365.2.1 element_count() . . . 2139
9.366 connext::Replier< TReq, TRep > Class Template Reference . . . 2139
9.366.1 Detailed Description . . . 2141
9.366.2 Member Typedef Documentation . . . 2142
9.366.2.1 Request . . . 2142
9.366.2.2 Reply . . . 2142
9.366.2.3 ReplyDataWriter . . . 2142
9.366.2.4 RequestDataReader . . . 2142
9.366.3 Constructor & Destructor Documentation . . . 2143
9.366.3.1 Replier() [1/2] . . . 2143
9.366.3.2 Replier() [2/2] . . . 2144
9.366.3.3 ~Replier() . . . 2144
9.366.4 Member Function Documentation . . . 2145
9.366.4.1 send_reply() [1/3] . . . 2145

9.366.4.2 send_reply() [2/3] . . . 2145
9.366.4.3 send_reply() [3/3] . . . 2146
9.366.4.4 receive_request() [1/2] . . . 2147
9.366.4.5 receive_request() [2/2] . . . 2147
9.366.4.6 receive_requests() [1/2] . . . 2147
9.366.4.7 receive_requests() [2/2] . . . 2148
9.366.4.8 wait_for_requests() [1/2] . . . 2148
9.366.4.9 wait_for_requests() [2/2] . . . 2148
9.366.4.10 take_request() [1/2] . . . 2149
9.366.4.11 take_request() [2/2] . . . 2149
9.366.4.12 take_requests() . . . 2150
9.366.4.13 read_request() [1/2] . . . 2150
9.366.4.14 read_request() [2/2] . . . 2150
9.366.4.15 read_requests() . . . 2150
9.366.4.16 get_request_datareader() . . . 2151
9.366.4.17 get_reply_datawriter() . . . 2151
9.367 connext::ReplierListener< TReq, TRep > Class Template Reference . . . 2151
9.367.1 Detailed Description . . . 2151
9.367.2 Member Function Documentation . . . 2152
9.367.2.1 on_request_available() . . . 2152
9.368 connext::ReplierParams< TReq, TRep > Class Template Reference . . . 2152
9.368.1 Detailed Description . . . 2153
9.368.2 Constructor & Destructor Documentation . . . 2153
9.368.2.1 ReplierParams() . . . 2153
9.368.3 Member Function Documentation . . . 2153
9.368.3.1 replier_listener() . . . 2154
9.368.3.2 service_name() . . . 2154
9.368.3.3 request_topic_name() . . . 2154
9.368.3.4 reply_topic_name() . . . 2154
9.368.3.5 qos_profile() . . . 2155
9.368.3.6 datawriter_qos() . . . 2155
9.368.3.7 datareader_qos() . . . 2155
9.368.3.8 publisher() . . . 2156
9.368.3.9 subscriber() . . . 2156
9.368.3.10 request_type_support() . . . 2156
9.368.3.11 reply_type_support() . . . 2157
9.369 connext::Requester< TReq, TRep > Class Template Reference . . . 2157
9.369.1 Detailed Description . . . 2159
9.369.2 Member Typedef Documentation . . . 2160

9.369.2.1 Request . . . 2160
9.369.2.2 Reply . . . 2161
9.369.2.3 RequestDataWriter . . . 2161
9.369.2.4 ReplyDataReader . . . 2161
9.369.2.5 RequestTypeSupport . . . 2161
9.369.2.6 ReplyTypeSupport . . . 2161
9.369.3 Constructor & Destructor Documentation . . . 2161
9.369.3.1 Requester() [1/2] . . . 2161
9.369.3.2 Requester() [2/2] . . . 2162
9.369.3.3 ∼Requester() . . . 2162
9.369.4 Member Function Documentation . . . 2163
9.369.4.1 send_request() [1/3] . . . 2163
9.369.4.2 send_request() [2/3] . . . 2163
9.369.4.3 send_request() [3/3] . . . 2164
9.369.4.4 receive_reply() [1/2] . . . 2165
9.369.4.5 receive_reply() [2/2] . . . 2165
9.369.4.6 receive_replies() [1/2] . . . 2166
9.369.4.7 receive_replies() [2/2] . . . 2166
9.369.4.8 wait_for_replies() [1/3] . . . 2167
9.369.4.9 wait_for_replies() [2/3] . . . 2168
9.369.4.10 wait_for_replies() [3/3] . . . 2168
9.369.4.11 take_reply() [1/4] . . . 2169
9.369.4.12 take_reply() [2/4] . . . 2170
9.369.4.13 take_replies() [1/3] . . . 2170
9.369.4.14 take_reply() [3/4] . . . 2171
9.369.4.15 take_reply() [4/4] . . . 2172
9.369.4.16 take_replies() [2/3] . . . 2173
9.369.4.17 take_replies() [3/3] . . . 2174
9.369.4.18 read_reply() [1/4] . . . 2175
9.369.4.19 read_reply() [2/4] . . . 2175
9.369.4.20 read_replies() [1/3] . . . 2175
9.369.4.21 read_reply() [3/4] . . . 2176
9.369.4.22 read_reply() [4/4] . . . 2176
9.369.4.23 read_replies() [2/3] . . . 2176
9.369.4.24 read_replies() [3/3] . . . 2177
9.369.4.25 get_request_datawriter() . . . 2177
9.369.4.26 get_reply_datareader() . . . 2178
9.370 connext::RequesterParams Class Reference . . . 2178
9.370.1 Detailed Description . . . 2179

9.370.2 Constructor & Destructor Documentation . . . 2179
9.370.2.1 RequesterParams() . . . 2179
9.370.3 Member Function Documentation . . . 2180
9.370.3.1 service_name() . . . 2180
9.370.3.2 request_topic_name() . . . 2180
9.370.3.3 reply_topic_name() . . . 2180
9.370.3.4 qos_profile() . . . 2181
9.370.3.5 datawriter_qos() . . . 2181
9.370.3.6 datareader_qos() . . . 2181
9.370.3.7 publisher() . . . 2182
9.370.3.8 subscriber() . . . 2182
9.370.3.9 request_type_support() . . . 2182
9.370.3.10 reply_type_support() . . . 2182
9.371 connext::RuntimeException Class Reference . . . 2183
9.371.1 Detailed Description . . . 2183
9.372 connext::Sample< T > Class Template Reference . . . 2183
9.372.1 Detailed Description . . . 2184
9.372.2 Constructor & Destructor Documentation . . . 2184
9.372.2.1 Sample() [1/3] . . . 2184
9.372.2.2 Sample() [2/3] . . . 2184
9.372.2.3 Sample() [3/3] . . . 2185
9.372.3 Member Function Documentation . . . 2185
9.372.3.1 identity() . . . 2185
9.372.3.2 related_identity() . . . 2185
9.372.3.3 data() [1/2] . . . 2186
9.372.3.4 info() [1/2] . . . 2186
9.372.3.5 data() [2/2] . . . 2186
9.372.3.6 info() [2/2] . . . 2187
9.373 rti::flat::Sample< OffsetType > Class Template Reference . . . 2187
9.373.1 Detailed Description . . . 2187
9.373.2 Member Typedef Documentation . . . 2188
9.373.2.1 Offset . . . 2188
9.373.2.2 ConstOffset . . . 2189
9.373.3 Member Function Documentation . . . 2189
9.373.3.1 root() [1/2] . . . 2189
9.373.3.2 root() [2/2] . . . 2189
9.373.3.3 create_data() . . . 2190
9.373.3.4 clone() . . . 2190
9.373.3.5 delete_data() . . . 2190

9.374 connext::SampleIterator< T, IsConst > Class Template Reference . . . 2191
9.374.1 Detailed Description . . . 2191
9.374.2 Member Function Documentation . . . 2191
9.374.2.1 operator++() . . . 2191
9.375 connext::SampleRef< T > Class Template Reference . . . 2191
9.375.1 Detailed Description . . . 2192
9.375.2 Constructor & Destructor Documentation . . . 2193
9.375.2.1 SampleRef() [1/4] . . . 2193
9.375.2.2 SampleRef() [2/4] . . . 2193
9.375.2.3 SampleRef() [3/4] . . . 2194
9.375.2.4 SampleRef() [4/4] . . . 2194
9.375.3 Member Function Documentation . . . 2194
9.375.3.1 operator=() . . . 2194
9.375.3.2 data() . . . 2195
9.375.3.3 info() . . . 2195
9.375.3.4 operator T&() . . . 2196
9.375.3.5 set_data() . . . 2196
9.375.3.6 set_info() . . . 2196
9.375.3.7 is_nil_data() . . . 2197
9.375.3.8 is_nil_info() . . . 2197
9.375.3.9 identity() . . . 2197
9.375.3.10 related_identity() . . . 2198
9.376 rti::flat::SequenceIterator< E, OffsetKind > Class Template Reference . . . 2198
9.376.1 Detailed Description . . . 2199
9.376.2 Member Typedef Documentation . . . 2200
9.376.2.1 iterator_category . . . 2200
9.376.2.2 value_type . . . 2200
9.376.2.3 reference . . . 2200
9.376.2.4 pointer . . . 2201
9.376.2.5 difference_type . . . 2201
9.376.3 Constructor & Destructor Documentation . . . 2201
9.376.3.1 SequenceIterator() . . . 2201
9.376.4 Member Function Documentation . . . 2201
9.376.4.1 is_null() . . . 2201
9.376.4.2 operator*() . . . 2202
9.376.4.3 operator->() . . . 2202
9.376.4.4 advance() . . . 2202
9.376.4.5 operator++() [1/2] . . . 2202
9.376.4.6 operator++() [2/2] . . . 2203

9.376.5 Friends And Related Function Documentation . . . 2203
9.376.5.1 operator< . . . 2203
9.376.5.2 operator> . . . 2203
9.376.5.3 operator<= . . . 2203
9.376.5.4 operator>= . . . 2204
9.376.5.5 operator== . . . 2204
9.376.5.6 operator"!= . . . 2204
9.377 rti::flat::SequenceOffset< ElementOffset > Class Template Reference . . . 2204
9.377.1 Detailed Description . . . 2205
9.377.2 Member Function Documentation . . . 2205
9.377.2.1 get_element() . . . 2205
9.377.2.2 element_count() . . . 2206
9.378 connext::SimpleReplier< TReq, TRep > Class Template Reference . . . 2206
9.378.1 Detailed Description . . . 2206
9.378.2 Constructor & Destructor Documentation . . . 2207
9.378.2.1 SimpleReplier() [1/2] . . . 2207
9.378.2.2 SimpleReplier() [2/2] . . . 2207
9.378.2.3 ~SimpleReplier() . . . 2207
9.379 connext::SimpleReplierListener< TReq, TRep > Class Template Reference . . . 2208
9.379.1 Detailed Description . . . 2208
9.379.2 Member Function Documentation . . . 2208
9.379.2.1 on_request_available() . . . 2208
9.379.2.2 return_loan() . . . 2209
9.380 connext::SimpleReplierParams< TReq, TRep > Class Template Reference . . . 2209
9.380.1 Detailed Description . . . 2210
9.380.2 Constructor & Destructor Documentation . . . 2210
9.380.2.1 SimpleReplierParams() . . . 2210
9.380.3 Member Function Documentation . . . 2211
9.380.3.1 service_name() . . . 2211
9.380.3.2 request_topic_name() . . . 2211
9.380.3.3 reply_topic_name() . . . 2211
9.380.3.4 qos_profile() . . . 2212
9.380.3.5 datawriter_qos() . . . 2212
9.380.3.6 datareader_qos() . . . 2212
9.380.3.7 publisher() . . . 2213
9.380.3.8 subscriber() . . . 2213
9.380.3.9 request_type_support() . . . 2213
9.380.3.10 reply_type_support() . . . 2214
9.381 rti::flat::StringBuilder Class Reference . . . 2214

9.381.1 Detailed Description . . . 2214
9.381.2 Member Function Documentation . . . 2215
9.381.2.1 set_string() . . . 2215
9.381.2.2 finish() . . . 2215
9.382 rti::flat::StringOffset Class Reference . . . 2216
9.382.1 Detailed Description . . . 2216
9.382.2 Member Function Documentation . . . 2216
9.382.2.1 get_string() [1/2] . . . 2216
9.382.2.2 get_string() [2/2] . . . 2217
9.382.2.3 element_count() . . . 2217
9.383 connext::TimeoutException Class Reference . . . 2217
9.383.1 Detailed Description . . . 2217
9.384 TransportAllocationSettings_t Struct Reference . . . 2218
9.384.1 Detailed Description . . . 2218
9.385 rti::flat::UnionBuilder< Discriminator > Class Template Reference . . . 2218
9.385.1 Detailed Description . . . 2219
9.386 connext::UnsupportedException Class Reference . . . 2219
9.386.1 Detailed Description . . . 2219
9.387 connext::WriteSample< T > Class Template Reference . . . 2219
9.387.1 Detailed Description . . . 2220
9.387.2 Constructor & Destructor Documentation . . . 2220
9.387.2.1 WriteSample() [1/4] . . . 2221
9.387.2.2 WriteSample() [2/4] . . . 2221
9.387.2.3 WriteSample() [3/4] . . . 2221
9.387.2.4 WriteSample() [4/4] . . . 2221
9.387.3 Member Function Documentation . . . 2221
9.387.3.1 data() [1/2] . . . 2222
9.387.3.2 info() [1/2] . . . 2222
9.387.3.3 data() [2/2] . . . 2222
9.387.3.4 info() [2/2] . . . 2222
9.387.3.5 identity() . . . 2223
9.388 connext::WriteSampleRef< T > Class Template Reference . . . 2223
9.388.1 Detailed Description . . . 2223
9.388.2 Constructor & Destructor Documentation . . . 2224
9.388.2.1 WriteSampleRef() [1/3] . . . 2224
9.388.2.2 WriteSampleRef() [2/3] . . . 2224
9.388.2.3 WriteSampleRef() [3/3] . . . 2224
9.388.3 Member Function Documentation . . . 2225
9.388.3.1 data() . . . 2225

9.388.3.2 info() . . . 2225
9.388.3.3 set_data() . . . 2225
9.388.3.4 set_info() . . . 2226
9.388.3.5 is_nil_data() . . . 2226
9.388.3.6 identity() . . . 2226

10 File Documentation **2227**

10.1 howto_prototypes.h . . . 2227
10.2 transport_common.h . . . 2228
10.3 transport_common_user.h . . . 2233
10.4 transport_interface_user.h . . . 2242
10.5 transport_shmem.h . . . 2245
10.6 transport_udp.h . . . 2249
10.7 transport_udpv4.h . . . 2262
10.8 transport_udpv6.h . . . 2273
10.9 AggregationBuilders.hpp . . . 2281
10.10 Builder.hpp . . . 2287
10.11 BuilderHelper.hpp . . . 2295
10.12 ExceptionHelper.hpp . . . 2298
10.13 FlatSample.hpp . . . 2300
10.14 FlatSampleImpl.hpp . . . 2303
10.15 FlatTypeTraits.hpp . . . 2304
10.16 Offset.hpp . . . 2305
10.17 rtiflat.hpp . . . 2314
10.18 SequenceBuilders.hpp . . . 2315
10.19 SequenceIterator.hpp . . . 2324
10.20 SequenceOffsets.hpp . . . 2327
10.21 Interpreter.hpp . . . 2334
10.22 Stream.hpp . . . 2338

11 Example Documentation **2345**

11.1 HelloWorld.idl . . . 2345
11.1.1 IDL Type Description . . . 2345
11.1.1.1 HelloWorld.idl . . . 2345
11.2 HelloWorld.cxx . . . 2346
11.2.1 Programming Language Type Description . . . 2346
11.2.1.1 HelloWorld.h . . . 2346
11.2.1.2 HelloWorld.cxx . . . 2347
11.3 HelloWorldSupport.cxx . . . 2352
11.3.1 User Data Type Support . . . 2352

11.3.1.1 HelloWorldSupport.h . . . 2352
11.3.1.2 HelloWorldSupport.cxx . . . 2352
11.4 HelloWorldPlugin.cxx . . . 2354
11.4.1 RTI Connext Implementation Support . . . 2354
11.4.1.1 HelloWorldPlugin.h . . . 2354
11.4.1.2 HelloWorldPlugin.cxx . . . 2356
11.5 HelloWorld_publisher.cxx . . . 2365
11.5.1 RTI Connext Publication Example . . . 2365
11.5.1.1 HelloWorld_publisher.cxx . . . 2365
11.6 HelloWorld_subscriber.cxx . . . 2367
11.6.1 RTI Connext Subscription Example . . . 2367
11.6.1.1 HelloWorld_subscriber.cxx . . . 2367

Index **2371**

Chapter 1

RTI Connext

Core Libraries and Utilities

Real-Time Innovations, Inc.

RTI Connext is network middleware for real-time distributed applications. It provides the communications services that programmers need to distribute time-critical data between embedded and/or enterprise devices or nodes. RTI Connext uses the publish-subscribe communications model to make data distribution efficient and robust.

The RTI Connext Application Programming Interface (API) is based on the OMG's Data Distribution Service (DDS) specification, version 1.4. The most recent publication of this specification can be found in the Catalog of OMG Specifications under "Platform Categories".

Note: This documentation is for the **RTI Connext Traditional C++ API**; the RTI Connext Modern C++ API is also available.

1.1 Available Documentation.

The documentation for this release is provided in two forms: the API Reference HTML documentation and PDF documents. If you are new to RTI Connext, the **Documentation Roadmap** (p. 430) will provide direction on how to learn about this product.

1.1.1 The documents for the Core Libraries and Utilities are:

- **What's New.** An overview of the new features in this release.
- **Release Notes.** System requirements, compatibility, what's fixed in this release, and known issues.
- **Platform Notes.** Specific details, such as compilation setting and libraries, related to building and using RTI Connext on the various supported platforms.
- **Getting Started Guide.** Core value and concepts behind the product, taking you step-by-step through the creation of a simple example application. Developers should read this document first.
- **Code Generator User's Manual.** Information about using rtiddsgen to generate code from data types.
- **User's Manual.** Introduction to RTI Connext, product tour and conceptual presentation of the functionality of RTI Connext.
- **QoS Reference Guide.** A compact summary of supported Quality of Service (QoS) policies.
- **XML-Based Application Creation Getting Started Guide.** Details on how to use XML-Based Application Creation.
- **Extensible Types Guide.** Additional information about extensible types.
- See more `documentation` on RTI Community.

1.1.2 The API Reference HTML documentation contains:

- **RTI Connext DDS API Reference** (p. 434) - The RTI Connext API reference.
- **Additional RTI Connext Communication Patterns** (p. 438) - RTI Connext APIs for additional communication patterns
- **Programming How-To's** (p. 439) - Describes and shows the common tasks done using the API.

The API Reference HTML documentation can be accessed through the tree view in the left frame of the web browser window. The bulk of the documentation is found under the entry labeled "Modules".

1.2 Feedback and Support for this Release.

We welcome any input on how to improve RTI Connext to suit your needs. If you have questions or comments about this release, please visit the RTI Customer Portal at `https://support.rti.com`.

The Customer Portal provides access to RTI software, documentation, and support. It also allows you to log support cases. Furthermore, the portal provides detailed solutions and a free public knowledge base. To access the software, documentation or log support cases, the RTI Customer Portal requires a username and password. You will receive this in the email confirming your purchase. If you do not have this email, please contact `license@rti.com`. Resetting your login password can be done directly at the RTI Customer Portal.

Chapter 2

Module Index

2.1 Modules

Here is a list of all modules:

Documentation Roadmap . . . 430
Conventions . . . 431
Using DDS:: Namespace . . . 433
RTI Connext DDS API Reference . . . 434
Domain Module . . . 41
DomainParticipantFactory . . . 42
DomainParticipantConfigParams . . . 716
DomainParticipants . . . 51
Built-in Topics . . . 61
Participant Built-in Topics . . . 490
Topic Built-in Topics . . . 492
Publication Built-in Topics . . . 493
Subscription Built-in Topics . . . 494
ServiceRequest Built-in Topic . . . 496
Common types and functions . . . 499
Topic Module . . . 63
Topics . . . 64
Zero Copy Transfer Over Shared Memory . . . 73
User Data Type Support . . . 73
Type Code Support . . . 79
Built-in Types . . . 95
String Built-in Type . . . 508
KeyedString Built-in Type . . . 509
Octets Built-in Type . . . 510
KeyedOctets Built-in Type . . . 511
Built-in Topic's Trust Types . . . 99
Dynamic Data . . . 103
DDS-Specific Primitive Types . . . 511
FlatData Topic-Types . . . 776
FlatData Builders . . . 766

FlatData Samples . . . 769
FlatData Offsets . . . 771
Publication Module . . . 107
Publishers . . . 107
Data Writers . . . 115
Flow Controllers . . . 122
Multi-channel DataWriters . . . 170
Subscription Module . . . 128
Subscribers . . . 130
DataReaders . . . 138
Read Conditions . . . 151
Query Conditions . . . 152
Topic Queries . . . 155
Data Samples . . . 152
Sample States . . . 160
View States . . . 162
Instance States . . . 164
Stream Kinds . . . 166
SampleProcessor . . . 756
Infrastructure Module . . . 167
Clock Selection . . . 40
Time Support . . . 517
GUID Support . . . 524
Sequence Number Support . . . 527
Exception Codes . . . 529
Return Codes . . . 531
Status Kinds . . . 533
QoS . . . 549
All QoS Policies . . . 549
ASYNCHRONOUS_PUBLISHER . . . 561
AVAILABILITY . . . 561
BATCH . . . 562
DATABASE . . . 563
DATA_READER_PROTOCOL . . . 564
DATA_READER_RESOURCE_LIMITS . . . 564
DATA_REPRESENTATION . . . 567
Compression Settings . . . 570
DATA_TAG . . . 574
DATA_WRITER_PROTOCOL . . . 579
DATA_WRITER_RESOURCE_LIMITS . . . 580
DATA_WRITER_TRANSFER_MODE . . . 583
DEADLINE . . . 583
DESTINATION_ORDER . . . 584
DISCOVERY . . . 586
NDDS_DISCOVERY_PEERS . . . 662
DISCOVERY_CONFIG . . . 587
DOMAIN_PARTICIPANT_RESOURCE_LIMITS . . . 594
DURABILITY . . . 596
DURABILITY_SERVICE . . . 600
ENTITY_FACTORY . . . 600
ENTITY_NAME . . . 601
EVENT . . . 602
HISTORY . . . 603

GROUP_DATA . . . 604
LATENCY_BUDGET . . . 605
LIFESPAN . . . 606
LIVELINESS . . . 607
LOCATORFILTER . . . 608
LOGGING . . . 609
MONITORING . . . 610
MULTICHANNEL . . . 611
OWNERSHIP . . . 612
OWNERSHIP_STRENGTH . . . 613
PARTITION . . . 614
PRESENTATION . . . 615
PROFILE . . . 616
PROPERTY . . . 617
PUBLISH_MODE . . . 627
READER_DATA_LIFECYCLE . . . 630
RECEIVER_POOL . . . 630
RELIABILITY . . . 631
RESOURCE_LIMITS . . . 635
SERVICE . . . 636
SYSTEM_RESOURCE_LIMITS . . . 638
TIME_BASED_FILTER . . . 639
TOPIC_DATA . . . 639
TOPIC_QUERY_DISPATCH . . . 640
TRANSPORT_BUILTIN . . . 641
TRANSPORT_MULTICAST . . . 645
Multicast Settings . . . 661
Multicast Mapping . . . 661
TRANSPORT_MULTICAST_MAPPING . . . 646
TRANSPORT_PRIORITY . . . 647
TRANSPORT_SELECTION . . . 648
TRANSPORT_UNICAST . . . 649
Unicast Settings . . . 660
TYPE_CONSISTENCY_ENFORCEMENT . . . 649
TYPESUPPORT . . . 651
USER_DATA . . . 653
WRITER_DATA_LIFECYCLE . . . 654
WIRE_PROTOCOL . . . 654
Extended Qos Support . . . 660
Thread Settings . . . 546
QoS by Category . . . 560
Basic QoS . . . 201
DDS Specification QoS . . . 209
DDS Extension QoS . . . 213
Functional Categories . . . 252
Application Acknowledgement . . . 253
Application-Defined Metadata . . . 254
Application Startup . . . 256
Batching . . . 257
Concurrency . . . 258
Content Filtering . . . 260
Data Availability . . . 261
Debuggability . . . 266
Discovery . . . 271

Discovery Isolation . . . 277
Durability . . . 278
Flow Control . . . 282
Fragmentation . . . 284
Instance Lifecycle . . . 286
Sending Large Data . . . 290
Liveliness . . . 292
Logging . . . 294
Middleware Memory Management . . . 297
QoS Profile Management . . . 312
Reliability QoS . . . 314
Sample Memory Management QoS . . . 321
Sample Ordering . . . 324
Security . . . 325
Throughput . . . 327
Transports . . . 327
Topic Queries . . . 330
Type System . . . 331
QoS Mutability . . . 333
Immutable QoS . . . 333
Mutable QoS . . . 369
Other QoS Attributes . . . 375
Applied Per-Instance . . . 375
Discovery Metadata . . . 377
Matching . . . 382
Deprecated QoS . . . 384
Entity Support . . . 669
Conditions and WaitSets . . . 670
AsyncWaitSet . . . 488
Cookie . . . 671
Sample Flags . . . 672
WriteParams . . . 674
Heap Support in C . . . 677
Builtin Qos Profiles . . . 679
User-managed Threads . . . 718
Octet Buffer Support . . . 753
Sequence Support . . . 757
Built-in Sequences . . . 169
String Support . . . 757
Transports . . . 172
Installing Transport Plugins . . . 177
Built-in Transport Plugins . . . 182
UDP Transport Plugin definitions . . . 454
Shared Memory Transport . . . 456
UDPv4 Transport . . . 462
Real-Time WAN Transport . . . 472
UDPv6 Transport . . . 479
Creating New Transport Plugins . . . 183
Transport Plugins Configuration . . . 441
Transport Address . . . 448
Queries and Filters Syntax . . . 184
Logging and Version . . . 189
Version . . . 721

Logging . . . 722
Activity Context . . . 729
General Utilities and Compliance Configuration . . . 190
Compliance Configuration . . . 734
Heap Monitoring . . . 746
Network Capture . . . 747
Other Utilities . . . 753
Observability Framework . . . 190
Monitoring Library 2.0 . . . 719
Durability and Persistence . . . 196
System Properties . . . 389
Configuring QoS Profiles with XML . . . 390
Additional RTI Connext Communication Patterns . . . 438
Request-Reply Pattern . . . 191
Requester . . . 191
Replier . . . 192
Infrastructure . . . 192
RTI Connext Exceptions . . . 39
Utilities . . . 195
Programming How-To's . . . 439
Publication Example . . . 392
Subscription Example . . . 393
Participant Use Cases . . . 394
Topic Use Cases . . . 396
FlowController Use Cases . . . 397
Publisher Use Cases . . . 399
DataWriter Use Cases . . . 400
Subscriber Use Cases . . . 401
DataReader Use Cases . . . 403
Entity Use Cases . . . 406
Waitset Use Cases . . . 408
Transport Use Cases . . . 409
Filter Use Cases . . . 412
Creating Custom Content Filters . . . 415
Large Data Use Cases . . . 418
Request-Reply Examples . . . 419
Interface . . . 440

Chapter 3

Namespace Index

3.1 Namespace List

Here is a list of all documented namespaces with brief descriptions:

connext
Namespace for Connext 779
rti
The RTI namespace 781
rti::flat
<<***extension***>> *(p. 432)* Support for **FlatData Topic-Types** (p. 776) 781

Chapter 4

Hierarchical Index

4.1 Class Hierarchy

This inheritance list is sorted roughly, but not completely, alphabetically:

rti::flat::AbstractBuilder . . . 787
rti::flat::AbstractListBuilder< detail::DHeaderGenerator > . . . 789
rti::flat::AbstractSequenceBuilder< detail::DHeaderGenerator > . . . 792
rti::flat::FinalSequenceBuilder< ElementOffset > . . . 1913
rti::flat::MutableSequenceBuilder< ElementBuilder > . . . 2012
rti::flat::MutableArrayBuilder< ElementBuilder, N > . . . 2008
rti::flat::AbstractListBuilder< detail::primitive_sequence_dheader_gen< T >::type > . . . 789
rti::flat::AbstractSequenceBuilder< detail::primitive_sequence_dheader_gen< T >::type > . . . 792
rti::flat::PrimitiveSequenceBuilder< T > . . . 2135
rti::flat::AbstractListBuilder< detail::primitive_sequence_dheader_gen< char >::type > . . . 789
rti::flat::AbstractSequenceBuilder< detail::primitive_sequence_dheader_gen< char >::type > . . . 792
rti::flat::PrimitiveSequenceBuilder< char > . . . 2135
rti::flat::StringBuilder . . . 2214
rti::flat::AbstractListBuilder< DHeaderGen > . . . 789
rti::flat::AbstractSequenceBuilder< DHeaderGen > . . . 792
rti::flat::AggregationBuilder . . . 793
rti::flat::UnionBuilder< int32_t > . . . 2218
MyFlatUnionBuilder . . . 2032
MyFlatMutableBuilder . . . 2018
rti::flat::UnionBuilder< Discriminator > . . . 2218
DDS_AcknowledgmentInfo . . . 794
DDS_AckResponseData_t . . . 796
DDS_AllocationSettings_t . . . 796
DDS_AnnotationParameterValue . . . 798
DDS_AsynchronousPublisherQosPolicy . . . 798
DDS_AsyncWaitSetProperty_t . . . 802
DDS_AvailabilityQosPolicy . . . 805
DDS_BatchQosPolicy . . . 810
DDS_BooleanSeq . . . 814
DDS_BuiltinTopicKey_t . . . 814

DDS_BuiltinTopicReaderResourceLimits_t . . . 815
DDS_ChannelSettings_t . . . 822
DDS_ChannelSettingsSeq . . . 824
DDS_CharSeq . . . 824
DDS_CoherentSetInfo_t . . . 824
DDS_CompressionSettings_t . . . 826
DDS_ContentFilterProperty_t . . . 828
DDS_Cookie_t . . . 830
DDS_CookieSeq . . . 830
DDS_DatabaseQosPolicy . . . 831
DDS_DataReaderCacheStatus . . . 836
DDS_DataReaderProtocolQosPolicy . . . 843
DDS_DataReaderProtocolStatus . . . 847
DDS_DataReaderQos . . . 858
DDS_DataReaderResourceLimitsInstanceReplacementSettings . . . 866
DDS_DataReaderResourceLimitsQosPolicy . . . 867
DDS_DataRepresentationIdSeq . . . 886
DDS_DataRepresentationQosPolicy . . . 886
DDS_DataTags . . . 889
DDS_DataWriterCacheStatus . . . 889
DDS_DataWriterProtocolQosPolicy . . . 892
DDS_DataWriterProtocolStatus . . . 897
DDS_DataWriterQos . . . 909
DDS_DataWriterResourceLimitsQosPolicy . . . 917
DDS_DataWriterShmemRefTransferModeSettings . . . 926
DDS_DataWriterTransferModeQosPolicy . . . 927
DDS_DeadlineQosPolicy . . . 928
DDS_DestinationOrderQosPolicy . . . 931
DDS_DiscoveryConfigQosPolicy . . . 934
DDS_DiscoveryQosPolicy . . . 960
DDS_DomainParticipantConfigParams_t . . . 964
DDS_DomainParticipantFactoryQos . . . 967
DDS_DomainParticipantProtocolStatus . . . 970
DDS_DomainParticipantQos . . . 971
DDS_DomainParticipantResourceLimitsQosPolicy . . . 977
DDS_DoubleSeq . . . 1004
DDS_DurabilityQosPolicy . . . 1005
DDS_DurabilityServiceQosPolicy . . . 1009
DDS_Duration_t . . . 1012
DDS_DynamicData . . . 1013
DDS_DynamicDataInfo . . . 1133
DDS_DynamicDataJsonParserProperties_t . . . 1134
DDS_DynamicDataMemberInfo . . . 1135
DDS_DynamicDataProperty_t . . . 1137
DDS_DynamicDataSeq . . . 1138
DDS_DynamicDataTypeProperty_t . . . 1138
DDS_DynamicDataTypeSerializationProperty_t . . . 1139
DDS_EndpointGroup_t . . . 1142
DDS_EndpointGroupSeq . . . 1142
DDS_EndpointTrustAlgorithmInfo . . . 1143
DDS_EndpointTrustInterceptorAlgorithmInfo . . . 1144
DDS_EndpointTrustProtectionInfo . . . 1144
DDS_EntityFactoryQosPolicy . . . 1145
DDS_EntityNameQosPolicy . . . 1147

DDS_EnumMember . . . 1149
DDS_EnumMemberSeq . . . 1150
DDS_EventQosPolicy . . . 1150
DDS_ExpressionProperty . . . 1152
DDS_FilterSampleInfo . . . 1153
DDS_FloatSeq . . . 1156
DDS_FlowControllerProperty_t . . . 1156
DDS_FlowControllerTokenBucketProperty_t . . . 1157
DDS_GroupDataQosPolicy . . . 1160
DDS_GUID_t . . . 1161
DDS_HistoryQosPolicy . . . 1162
DDS_InconsistentTopicStatus . . . 1165
DDS_InstanceHandleSeq . . . 1166
DDS_Int8Seq . . . 1167
DDS_InvalidLocalAdvanceNoticeStatus . . . 1167
DDS_KeyedOctets . . . 1168
DDS_KeyedOctetsSeq . . . 1170
DDS_KeyedString . . . 1171
DDS_KeyedStringSeq . . . 1173
DDS_LatencyBudgetQosPolicy . . . 1173
DDS_LifespanQosPolicy . . . 1175
DDS_LivelinessChangedStatus . . . 1176
DDS_LivelinessLostStatus . . . 1179
DDS_LivelinessQosPolicy . . . 1180
DDS_Locator_t . . . 1184
DDS_LocatorFilter_t . . . 1185
DDS_LocatorFilterQosPolicy . . . 1186
DDS_LocatorFilterSeq . . . 1187
DDS_LocatorSeq . . . 1188
DDS_LoggingQosPolicy . . . 1188
DDS_LongDoubleSeq . . . 1192
DDS_LongLongSeq . . . 1192
DDS_LongSeq . . . 1192
DDS_MonitoringDedicatedParticipantSettings . . . 1193
DDS_MonitoringDistributionSettings . . . 1195
DDS_MonitoringEventDistributionSettings . . . 1197
DDS_MonitoringLoggingDistributionSettings . . . 1200
DDS_MonitoringLoggingForwardingSettings . . . 1204
DDS_MonitoringMetricSelection . . . 1206
DDS_MonitoringMetricSelectionSeq . . . 1208
DDS_MonitoringPeriodicDistributionSettings . . . 1209
DDS_MonitoringQosPolicy . . . 1210
DDS_MonitoringTelemetryData . . . 1213
DDS_MultiChannelQosPolicy . . . 1214
DDS_Octets . . . 1217
DDS_OctetSeq . . . 1218
DDS_OctetsSeq . . . 1219
DDS_OfferedDeadlineMissedStatus . . . 1219
DDS_OfferedIncompatibleQosStatus . . . 1221
DDS_OwnershipQosPolicy . . . 1222
DDS_OwnershipStrengthQosPolicy . . . 1227
DDS_ParticipantBuiltinTopicData . . . 1229
DDS_ParticipantBuiltinTopicDataSeq . . . 1235
DDS_ParticipantTrustAlgorithmInfo . . . 1235

DDS_ParticipantTrustInterceptorAlgorithmInfo . . . 1236
DDS_ParticipantTrustKeyEstablishmentAlgorithmInfo . . . 1237
DDS_ParticipantTrustProtectionInfo . . . 1238
DDS_ParticipantTrustSignatureAlgorithmInfo . . . 1239
DDS_PartitionQosPolicy . . . 1240
DDS_PersistentStorageSettings . . . 1242
DDS_PresentationQosPolicy . . . 1249
DDS_PrintFormatProperty . . . 1253
DDS_ProductVersion_t . . . 1255
DDS_ProfileQosPolicy . . . 1257
DDS_Property_t . . . 1259
DDS_PropertyQosPolicy . . . 1260
DDS_PropertySeq . . . 1262
DDS_ProtocolVersion_t . . . 1263
DDS_PublicationBuiltinTopicData . . . 1264
DDS_PublicationBuiltinTopicDataSeq . . . 1273
DDS_PublicationMatchedStatus . . . 1273
DDS_PublisherQos . . . 1276
DDS_PublishModeQosPolicy . . . 1279
DDS_QosPolicyCount . . . 1283
DDS_QosPolicyCountSeq . . . 1284
DDS_QosPrintAll_t . . . 1285
DDS_QosPrintFormat . . . 1285
DDS_QueryConditionParams . . . 1287
DDS_ReadConditionParams . . . 1288
DDS_ReaderDataLifecycleQosPolicy . . . 1289
DDS_ReceiverPoolQosPolicy . . . 1293
DDS_ReliabilityQosPolicy . . . 1295
DDS_ReliableReaderActivityChangedStatus . . . 1300
DDS_ReliableWriterCacheChangedStatus . . . 1302
DDS_ReliableWriterCacheEventCount . . . 1305
DDS_RequestedDeadlineMissedStatus . . . 1306
DDS_RequestedIncompatibleQosStatus . . . 1307
DDS_ResourceLimitsQosPolicy . . . 1308
DDS_RTPS_EntityId_t . . . 1313
DDS_RTPS_GUID_t . . . 1314
DDS_RtpsReliableReaderProtocol_t . . . 1314
DDS_RtpsReliableWriterProtocol_t . . . 1320
DDS_RtpsWellKnownPorts_t . . . 1338
DDS_SampleIdentity_t . . . 1343
DDS_SampleInfo . . . 1344
DDS_SampleInfoSeq . . . 1355
DDS_SampleLostStatus . . . 1355
DDS_SampleRejectedStatus . . . 1356
DDS_SequenceNumber_t . . . 1357
DDS_ServiceQosPolicy . . . 1358
DDS_ServiceRequest . . . 1359
DDS_ServiceRequestAcceptedStatus . . . 1361
DDS_ServiceRequestSeq . . . 1363
DDS_ShortSeq . . . 1363
DDS_StringSeq . . . 1364
DDS_StructMember . . . 1365
DDS_StructMemberSeq . . . 1367
DDS_SubscriberQos . . . 1367

DDS_SubscriptionBuiltinTopicData . . . 1370
DDS_SubscriptionBuiltinTopicDataSeq . . . 1379
DDS_SubscriptionMatchedStatus . . . 1379
DDS_SystemResourceLimitsQosPolicy . . . 1382
DDS_Tag . . . 1384
DDS_TagSeq . . . 1385
DDS_ThreadSettings_t . . . 1385
DDS_Time_t . . . 1387
DDS_TimeBasedFilterQosPolicy . . . 1389
DDS_TopicBuiltinTopicData . . . 1391
DDS_TopicBuiltinTopicDataSeq . . . 1396
DDS_TopicDataQosPolicy . . . 1396
DDS_TopicQos . . . 1397
DDS_TopicQueryData . . . 1402
DDS_TopicQueryDispatchQosPolicy . . . 1404
DDS_TopicQuerySelection . . . 1406
DDS_TransportBuiltinQosPolicy . . . 1408
DDS_TransportInfo_t . . . 1409
DDS_TransportInfoSeq . . . 1410
DDS_TransportMulticastMapping_t . . . 1410
DDS_TransportMulticastMappingFunction_t . . . 1412
DDS_TransportMulticastMappingQosPolicy . . . 1413
DDS_TransportMulticastMappingSeq . . . 1415
DDS_TransportMulticastQosPolicy . . . 1415
DDS_TransportMulticastSettings_t . . . 1417
DDS_TransportMulticastSettingsSeq . . . 1419
DDS_TransportPriorityQosPolicy . . . 1419
DDS_TransportSelectionQosPolicy . . . 1421
DDS_TransportUnicastQosPolicy . . . 1422
DDS_TransportUnicastSettings_t . . . 1424
DDS_TransportUnicastSettingsSeq . . . 1426
DDS_TrustAlgorithmRequirements . . . 1426
DDS_TypeAllocationParams_t . . . 1427
DDS_TypeCode . . . 1428
DDS_TypeCodeFactory . . . 1476
DDS_TypeCodePrintFormatProperty . . . 1490
DDS_TypeConsistencyEnforcementQosPolicy . . . 1493
DDS_TypeDeallocationParams_t . . . 1497
DDS_TypeSupportQosPolicy . . . 1499
DDS_UInt8Seq . . . 1500
DDS_UnionMember . . . 1501
DDS_UnionMemberSeq . . . 1502
DDS_UnsignedLongLongSeq . . . 1503
DDS_UnsignedLongSeq . . . 1503
DDS_UnsignedShortSeq . . . 1503
DDS_UserDataQosPolicy . . . 1504
DDS_ValueMember . . . 1505
DDS_ValueMemberSeq . . . 1508
DDS_VendorId_t . . . 1508
DDS_VirtualWriterInfo . . . 1509
DDS_WaitSetProperty_t . . . 1509
DDS_WcharSeq . . . 1510
DDS_WireProtocolQosPolicy . . . 1511
DDS_WriteParams_t . . . 1519

DDS_WriterDataLifecycleQosPolicy . . . 1524
DDS_WstringSeq . . . 1527
DDSAsyncWaitSet . . . 1528
DDSAsyncWaitSetCompletionToken . . . 1541
DDSAsyncWaitSetListener . . . 1543
DDSCondition . . . 1544
  DDSGuardCondition . . . 1739
  DDSReadCondition . . . 1842
    DDSQueryCondition . . . 1841
  DDSStatusCondition . . . 1846
DDSConditionHandler . . . 1546
  DDSDataReaderStatusConditionHandler . . . 1586
DDSConditionSeq . . . 1547
DDSContentFilter . . . 1548
  DDSWriterContentFilter . . . 1905
DDSDataReaderSeq . . . 1585
DDSDataTagQosPolicyHelper . . . 1588
DDSDomainParticipantFactory . . . 1694
DDSEntity . . . 1731
  DDSDomainEntity . . . 1618
    DDSDataReader . . . 1556
      DDSDynamicDataReader . . . 1724
      DDSDynamicDataReader . . . 1724
      DDSKeyedOctetsDataReader . . . 1741
      DDSKeyedStringDataReader . . . 1769
      DDSOctetsDataReader . . . 1800
      DDSParticipantBuiltinTopicDataDataReader . . . 1816
      DDSPublicationBuiltinTopicDataDataReader . . . 1817
      DDSServiceRequestDataReader . . . 1845
      DDSStringDataReader . . . 1848
      DDSSubscriptionBuiltinTopicDataDataReader . . . 1882
      DDSTopicBuiltinTopicDataDataReader . . . 1890
      FooDataReader . . . 1917
    DDSDataWriter . . . 1589
      DDSDynamicDataWriter . . . 1730
      DDSDynamicDataWriter . . . 1730
      DDSKeyedOctetsDataWriter . . . 1750
      DDSKeyedStringDataWriter . . . 1778
      DDSOctetsDataWriter . . . 1804
      DDSStringDataWriter . . . 1852
      FooDataWriter . . . 1944
    DDSPublisher . . . 1818
    DDSSubscriber . . . 1860
    DDSTopic . . . 1885
  DDSDomainParticipant . . . 1619
DDSFlowController . . . 1736
DDSListener . . . 1794
  DDSDataReaderListener . . . 1583
    DDSSubscriberListener . . . 1881
      DDSDomainParticipantListener . . . 1722
  DDSDataWriterListener . . . 1612
    DDSPublisherListener . . . 1839

DDSDomainParticipantListener . . . 1722
DDSTopicListener . . . 1894
DDSDomainParticipantListener . . . 1722
DDSParticipantBuiltinTopicDataTypeSupport . . . 1816
DDSPropertyQosPolicyHelper . . . 1817
DDSPublicationBuiltinTopicDataTypeSupport . . . 1818
DDSPublisherSeq . . . 1840
DDSServiceRequestTypeSupport . . . 1845
DDSSubscriberSeq . . . 1882
DDSSubscriptionBuiltinTopicDataTypeSupport . . . 1883
DDSThreadFactory . . . 1883
DDSTopicBuiltinTopicDataTypeSupport . . . 1891
DDSTopicDescription . . . 1892
DDSContentFilteredTopic . . . 1551
DDSMultiTopic . . . 1797
DDSTopic . . . 1885
DDSTopicQuery . . . 1895
DDSTopicQueryHelper . . . 1896
DDSTypeSupport . . . 1897
DDSDynamicDataTypeSupport . . . 1725
DDSKeyedOctetsTypeSupport . . . 1763
DDSKeyedStringTypeSupport . . . 1788
DDSOctetsTypeSupport . . . 1811
DDSStringTypeSupport . . . 1855
FooTypeSupport . . . 1979
DDSWaitSet . . . 1897
rti::flat::flat_type_traits< T > . . . 1915
rti::flat::flat_type_traits< PrimitiveSequenceOffset< char > > . . . 1915
Foo . . . 1916
FooSeq . . . 1965
connext::IsInvalidSamplePredicate< T > . . . 1993
connext::IsReplyRelatedPredicate< T > . . . 1993
connext::IsValidSamplePredicate< T > . . . 1995
connext::LoanedSamples< T > . . . 1995
connext::LoanedSamples< T >::LoanMemento . . . 2004
connext::LogicException . . . 2005
connext::AlreadyDeletedException . . . 793
connext::BadParameterException . . . 794
connext::IllegalOperationException . . . 1992
connext::ImmutablePolicyException . . . 1992
connext::InconsistentPolicyException . . . 1993
connext::NotEnabledException . . . 2126
connext::PreconditionNotMetException . . . 2132
connext::UnsupportedException . . . 2219
connext::MessagingLibraryVersion . . . 2005
connext::MessagingVersion . . . 2006
NDDS_Config_LibraryVersion_t . . . 2039
NDDS_Config_LogMessage . . . 2040
NDDS_Transport_Address_t . . . 2042
NDDS_Transport_Interface_t . . . 2043
NDDS_Transport_Property_t . . . 2044
NDDS_Transport_Shmem_Property_t . . . 2052
NDDS_Transport_UDP_WAN_CommPortsMappingInfo . . . 2055

NDDS_Transport_UDPv4_Property_t . . . 2056
NDDS_Transport_UDPv4_WAN_Property_t . . . 2067
NDDS_Transport_UDPv6_Property_t . . . 2077
NDDS_Transport_UUID . . . 2087
NDDS_Utility_NetworkCaptureParams_t . . . 2087
NDDSConfigActivityContext . . . 2089
NDDSConfigCompliance . . . 2090
NDDSConfigLogger . . . 2093
NDDSConfigLoggerDevice . . . 2100
NDDSConfigVersion . . . 2101
NDDSLM . . . 2103
NDDSTransportSupport . . . 2104
NDDSUtility . . . 2110
NDDSUtilityHeapMonitoring . . . 2113
NDDSUtilityNetworkCapture . . . 2117
rti::flat::OffsetBase . . . 2126
  rti::flat::AbstractPrimitiveList< char > . . . 790
    rti::flat::PrimitiveSequenceOffset< char > . . . 2138
      rti::flat::StringOffset . . . 2216
  rti::flat::FinalOffset< MyFlatFinalOffset > . . . 1913
    MyFlatFinalOffset . . . 2014
  rti::flat::AbstractAlignedList< ElementOffset > . . . 785
    rti::flat::FinalAlignedArrayOffset< ElementOffset, N > . . . 1909
    rti::flat::MutableArrayOffset< ElementOffset, N > . . . 2010
    rti::flat::SequenceOffset< ElementOffset > . . . 2204
  rti::flat::AbstractPrimitiveList< T > . . . 790
    rti::flat::PrimitiveArrayOffset< T, N > . . . 2132
    rti::flat::PrimitiveSequenceOffset< T > . . . 2138
  rti::flat::FinalArrayOffset< ElementOffset, N > . . . 1911
  rti::flat::FinalOffset< T > . . . 1913
  rti::flat::MutableOffset . . . 2011
    MyFlatMutableOffset . . . 2025
    MyFlatUnionOffset . . . 2035
  rti::flat::PrimitiveConstOffset< T > . . . 2133
    rti::flat::PrimitiveOffset< T > . . . 2134
connext::Replier< TReq, TRep > . . . 2139
connext::ReplierListener< TReq, TRep > . . . 2151
connext::ReplierParams< TReq, TRep > . . . 2152
connext::Requester< TReq, TRep > . . . 2157
connext::RequesterParams . . . 2178
connext::RuntimeException . . . 2183
  connext::OutOfResourcesException . . . 2131
  connext::TimeoutException . . . 2217
connext::Sample< T > . . . 2183
rti::flat::Sample< OffsetType > . . . 2187
connext::SampleIterator< T, IsConst > . . . 2191
connext::SampleRef< T > . . . 2191
rti::flat::SequenceIterator< E, OffsetKind > . . . 2198
connext::SimpleReplier< TReq, TRep > . . . 2206
connext::SimpleReplierListener< TReq, TRep > . . . 2208
connext::SimpleReplierParams< TReq, TRep > . . . 2209
TransportAllocationSettings_t . . . 2218
connext::WriteSample< T > . . . 2219
connext::WriteSampleRef< T > . . . 2223

Chapter 5

Class Index

5.1 Class List

Here are the classes, structs, unions and interfaces with brief descriptions:

rti::flat::AbstractAlignedList< ElementOffset >
Base class of Offsets to sequences and arrays of non-primitive members . . . 785
rti::flat::AbstractBuilder
Base class of all **Builders** (p. 766) . . . 787
rti::flat::AbstractListBuilder< DHeaderGen >
Base class of all array and sequence builders . . . 789
rti::flat::AbstractPrimitiveList< T >
Base class for Offsets to sequences and arrays of primitive types . . . 790
rti::flat::AbstractSequenceBuilder< DHeaderGen >
Base class of Builders for sequence members . . . 792
rti::flat::AggregationBuilder
Base class of struct and union builders . . . 793
connext::AlreadyDeletedException
The object target of this operation has already been deleted . . . 793
connext::BadParameterException
Illegal parameter value . . . 794
DDS_AcknowledgmentInfo
Information about an application-level acknowledged sample . . . 794
DDS_AckResponseData_t
Data payload of an application-level acknowledgment . . . 796
DDS_AllocationSettings_t
Resource allocation settings . . . 796
DDS_AnnotationParameterValue
Annotation parameter value . . . 798
DDS_AsynchronousPublisherQosPolicy
<<***extension***>> *(p. 213)* Configures the mechanism that sends user data in an external middleware thread . . . 798
DDS_AsyncWaitSetProperty_t
Specifies the **DDSAsyncWaitSet** (p. 1528) behavior . . . 802
DDS_AvailabilityQosPolicy
<<***extension***>> *(p. 213)* Configures the availability of data . . . 805

DDS_BatchQosPolicy
<<***extension***>> *(p. 213)* Used to configure batching of multiple samples into a single network packet in order to increase throughput for small samples . . . 810
DDS_BooleanSeq
Instantiates **FooSeq** (p. 1965) < **DDS_Boolean** (p. 516) > . . . 814
DDS_BuiltinTopicKey_t
The key type of the built-in topic types . . . 814
DDS_BuiltinTopicReaderResourceLimits_t
<<***extension***>> *(p. 213)* Built-in topic reader's resource limits . . . 815
DDS_ChannelSettings_t
<<***extension***>> *(p. 213)* Type used to configure the properties of a channel . . . 822
DDS_ChannelSettingsSeq
Declares IDL sequence< **DDS_ChannelSettings_t** (p. 822) > . . . 824
DDS_CharSeq
Instantiates **FooSeq** (p. 1965) < **DDS_Char** (p. 513) > . . . 824
DDS_CoherentSetInfo_t
<<***extension***>> *(p. 432)* Type definition for a coherent set info . . . 824
DDS_CompressionSettings_t
<<***extension***>> *(p. 213)* Settings related to compressing user data . . . 826
DDS_ContentFilterProperty_t
<<***extension***>> *(p. 432)* Type used to provide all the required information to enable content filtering 828
DDS_Cookie_t
<<***extension***>> *(p. 432)* Sequence of bytes . . . 830
DDS_CookieSeq
Declares IDL sequence < **DDS_Cookie_t** (p. 830) > . . . 830
DDS_DatabaseQosPolicy
<<***extension***>> *(p. 213)* Various threads and resource limits settings used by RTI Connext to control its internal database . . . 831
DDS_DataReaderCacheStatus
<<***extension***>> *(p. 432)* The status of the reader's cache . . . 836
DDS_DataReaderProtocolQosPolicy
<<***extension***>> *(p. 213)* Along with **DDS_WireProtocolQosPolicy** (p. 1511) and **DDS_DataWriterProtocolQosPolicy** (p. 892), this QoS policy configures the DDS on-the-network protocol (RTPS) . . . 843
DDS_DataReaderProtocolStatus
<<***extension***>> *(p. 432)* The status of a reader's internal protocol related metrics, like the number of samples received, filtered, rejected; and status of wire protocol traffic . . . 847
DDS_DataReaderQos
QoS policies supported by a **DDSDataReader** (p. 1556) entity . . . 858
DDS_DataReaderResourceLimitsInstanceReplacementSettings
Instance replacement kind applied to each instance state . . . 866
DDS_DataReaderResourceLimitsQosPolicy
<<***extension***>> *(p. 213)* Various settings that configure how a **DDSDataReader** (p. 1556) allocates and uses physical memory for internal resources . . . 867
DDS_DataRepresentationIdSeq
Declares IDL sequence < **DDS_DataRepresentationId_t** (p. 568) > . . . 886
DDS_DataRepresentationQosPolicy
<<***spec***>> *(p. 209)* This QoS policy contains a list of representation identifiers and compression settings used by **DDSDataWriter** (p. 1589) and **DDSDataReader** (p. 1556) entities to negotiate which data representation and compression settings to use . . . 886
DDS_DataTags
Definition of **DDS_DataTagQosPolicy** (p. 575) . . . 889

DDS_DataWriterCacheStatus
<<***extension***>> *(p. 432)* The status of the DataWriter's cache. Provides information on cache related metrics such as the number of samples and instances in the DataWriter queue 889
DDS_DataWriterProtocolQosPolicy
<<***extension***>> *(p. 213)* Protocol that applies only to **DDSDataWriter** (p. 1589) instances 892
DDS_DataWriterProtocolStatus
<<***extension***>> *(p. 432)* The status of a writer's internal protocol related metrics, like the number of samples pushed, pulled, filtered; and status of wire protocol traffic 897
DDS_DataWriterQos
QoS policies supported by a **DDSDataWriter** (p. 1589) entity 909
DDS_DataWriterResourceLimitsQosPolicy
<<***extension***>> *(p. 213)* Various settings that configure how a **DDSDataWriter** (p. 1589) allocates and uses physical memory for internal resources 917
DDS_DataWriterShmemRefTransferModeSettings
Settings related to transferring data using shared memory references 926
DDS_DataWriterTransferModeQosPolicy
<<***extension***>> *(p. 213)* Qos related to transferring data 927
DDS_DeadlineQosPolicy
<<***spec***>> *(p. 209)* Expresses the maximum duration (deadline) within which an instance is expected to be updated 928
DDS_DestinationOrderQosPolicy
<<***spec***>> *(p. 209)* Controls how the middleware will deal with data sent by multiple **DDSDataWriter** (p. 1589) entities for the same instance of data (i.e., same **DDSTopic** (p. 1885) and key) . . . 931
DDS_DiscoveryConfigQosPolicy
Settings for discovery configuration 934
DDS_DiscoveryQosPolicy
<<***extension***>> *(p. 213)* Configures the mechanism used by the middleware to automatically discover and connect with new remote applications 960
DDS_DomainParticipantConfigParams_t
<<***extension***>> *(p. 432)* Input paramaters for creating a participant from configuration. It allows to modify or override some of the properties of the entities defined in the configuration 964
DDS_DomainParticipantFactoryQos
QoS policies supported by a **DDSDomainParticipantFactory** (p. 1694) 967
DDS_DomainParticipantProtocolStatus
<<***extension***>> *(p. 432)* The status of a participant's protocol related metrics, like the number of corrupted messages, change in corrupted messages and timestamp of the last corrupted message . 970
DDS_DomainParticipantQos
QoS policies supported by a **DDSDomainParticipant** (p. 1619) entity 971
DDS_DomainParticipantResourceLimitsQosPolicy
<<***extension***>> *(p. 213)* Various settings that configure how a **DDSDomainParticipant** (p. 1619) allocates and uses physical memory for internal resources, including the maximum sizes of various properties 977
DDS_DoubleSeq
Instantiates **FooSeq** (p. 1965) < **DDS_Double** (p. 515) > 1004
DDS_DurabilityQosPolicy
<<***spec***>> *(p. 209)* This QoS policy specifies whether or not RTI Connext will store and deliver previously published data samples to new **DDSDataReader** (p. 1556) entities that join the network later 1005
DDS_DurabilityServiceQosPolicy
<<***spec***>> *(p. 209)* Various settings to configure the external *RTI Persistence Service* used by RTI Connext for DataWriters with a **DDS_DurabilityQosPolicy** (p. 1005) setting of **DDS_PERSISTENT_DURABILITY_QOS** (p. 599) or **DDS_TRANSIENT_DURABILITY_QOS** (p. 599) 1009
DDS_Duration_t
Type for *duration* representation 1012

DDS_DynamicData
A sample of any complex data type, which can be inspected and manipulated reflectively 1013
DDS_DynamicDataInfo
A descriptor for a **DDS_DynamicData** (p. 1013) object 1133
DDS_DynamicDataJsonParserProperties_t
A collection of attributes used to configure **DDS_DynamicData** (p. 1013) objects 1134
DDS_DynamicDataMemberInfo
A descriptor for a single member (i.e. field) of dynamically defined data type 1135
DDS_DynamicDataProperty_t
A collection of attributes used to configure **DDS_DynamicData** (p. 1013) objects 1137
DDS_DynamicDataSeq
An ordered collection of **DDS_DynamicData** (p. 1013) elements 1138
DDS_DynamicDataTypeProperty_t
A collection of attributes used to configure **DDSDynamicDataTypeSupport** (p. 1725) objects . . . 1138
DDS_DynamicDataTypeSerializationProperty_t
Properties that govern how data of a certain type will be serialized on the network 1139
DDS_EndpointGroup_t
Specifies a group of endpoints that can be collectively identified by a name and satisfied by a quorum 1142
DDS_EndpointGroupSeq
A sequence of **DDS_EndpointGroup_t** (p. 1142) 1142
DDS_EndpointTrustAlgorithmInfo
Trust Plugins algorithm information associated with the discovered endpoint 1143
DDS_EndpointTrustInterceptorAlgorithmInfo
Trust Plugins interception algorithm information associated with the discovered endpoint 1144
DDS_EndpointTrustProtectionInfo
Trust Plugins Protection information associated with the discovered endpoint 1144
DDS_EntityFactoryQosPolicy
<<***spec***>> *(p. 209)* A QoS policy for all **DDSEntity** (p. 1731) types that can act as factories for one or more other **DDSEntity** (p. 1731) types 1145
DDS_EntityNameQosPolicy
<<***extension***>> *(p. 213)* Assigns a name and a role name to a **DDSDomainParticipant** (p. 1619), **DDSPublisher** (p. 1818), **DDSSubscriber** (p. 1860), **DDSDataWriter** (p. 1589) or **DDSDataReader** (p. 1556). Except for **DDSPublisher** (p. 1818) and **DDSSubscriber** (p. 1860), these names will be visible during the discovery process and in RTI tools to help you visualize and debug your system . 1147
DDS_EnumMember
A description of a member of an enumeration 1149
DDS_EnumMemberSeq
Defines a sequence of enumerator members 1150
DDS_EventQosPolicy
<<***extension***>> *(p. 213)* Settings for event 1150
DDS_ExpressionProperty
Provides additional information about the filter expression passed to **DDSWriterContentFilter**↩ **::writer_compile** (p. 1906) 1152
DDS_FilterSampleInfo
Provides meta information associated with the sample 1153
DDS_FloatSeq
Instantiates **FooSeq** (p. 1965) < **DDS_Float** (p. 515) > 1156
DDS_FlowControllerProperty_t
Determines the flow control characteristics of the **DDSFlowController** (p. 1736) 1156
DDS_FlowControllerTokenBucketProperty_t
DDSFlowController (p. 1736) uses the popular token bucket approach for open loop network flow control. The flow control characteristics are determined by the token bucket properties 1157

DDS_GroupDataQosPolicy
Attaches a buffer of opaque data that is distributed by means of **Built-in Topics** (p. 61) during discovery . . . 1160
DDS_GUID_t
Type for *GUID* (Global Unique Identifier) representation . . . 1161
DDS_HistoryQosPolicy
<<***spec***>> *(p. 209)* Specifies the behavior of RTI Connext in the case where the value of a sample changes (one or more times) before it can be successfully communicated to one or more existing subscribers . . . 1162
DDS_InconsistentTopicStatus
DDS_INCONSISTENT_TOPIC_STATUS (p. 539) . . . 1165
DDS_InstanceHandleSeq
Instantiates **FooSeq** (p. 1965) < **DDS_InstanceHandle_t** (p. 76) > . . . 1166
DDS_Int8Seq
Instantiates **FooSeq** (p. 1965) < **DDS_Int8** (p. 514) > . . . 1167
DDS_InvalidLocalAdvanceNoticeStatus
DDS_INVALID_LOCAL_ADVANCE_NOTICE_STATUS (p. 543) . . . 1167
DDS_KeyedOctets
Built-in type consisting of a variable-length array of opaque bytes and a string that is the key . . . 1168
DDS_KeyedOctetsSeq
Instantiates **FooSeq** (p. 1965) < **DDS_KeyedOctets** (p. 1168) > . . . 1170
DDS_KeyedString
Keyed string built-in type . . . 1171
DDS_KeyedStringSeq
Instantiates **FooSeq** (p. 1965) < **DDS_KeyedString** (p. 1171) > . . . 1173
DDS_LatencyBudgetQosPolicy
<<***spec***>> *(p. 209)* Provides a hint as to the maximum acceptable delay from the time the data is written to the time it is received by the subscribing applications . . . 1173
DDS_LifespanQosPolicy
<<***spec***>> *(p. 209)* Specifies how long the data written by the **DDSDataWriter** (p. 1589) is considered valid . . . 1175
DDS_LivelinessChangedStatus
DDS_LIVELINESS_CHANGED_STATUS (p. 542) . . . 1176
DDS_LivelinessLostStatus
DDS_LIVELINESS_LOST_STATUS (p. 542) . . . 1179
DDS_LivelinessQosPolicy
Specifies and configures the mechanism that allows **DDSDataReader** (p. 1556) entities to detect when **DDSDataWriter** (p. 1589) entities become disconnected or "dead." . . . 1180
DDS_Locator_t
<<***extension***>> *(p. 213)* Type used to represent the addressing information needed to send a message to an RTPS Endpoint using one of the supported transports . . . 1184
DDS_LocatorFilter_t
<<***extension***>> *(p. 213)* Specifies the configuration of an individual channel within a MultiChannel DataWriter . . . 1185
DDS_LocatorFilterQosPolicy
<<***extension***>> *(p. 213)* The QoS policy used to report the configuration of a MultiChannel DataWriter as part of **DDS_PublicationBuiltinTopicData** (p. 1264) . . . 1186
DDS_LocatorFilterSeq
Declares IDL sequence< **DDS_LocatorFilter_t** (p. 1185) > . . . 1187
DDS_LocatorSeq
Declares IDL sequence < **DDS_Locator_t** (p. 1184) > . . . 1188
DDS_LoggingQosPolicy
<<***extension***>> *(p. 213)* Configures the RTI Connext logging facility . . . 1188

DDS_LongDoubleSeq
Instantiates **FooSeq** (p. 1965) < **DDS_LongDouble** (p. 516) > . . . 1192
DDS_LongLongSeq
Instantiates **FooSeq** (p. 1965) < **DDS_LongLong** (p. 515) > . . . 1192
DDS_LongSeq
Instantiates **FooSeq** (p. 1965) < **DDS_Long** (p. 515) > . . . 1192
DDS_MonitoringDedicatedParticipantSettings
<<***extension***>> *(p. 213)* Configures the use of a dedicated **DDSDomainParticipant** (p. 1619) to distribute the RTI Connext application telemetry data . . . 1193
DDS_MonitoringDistributionSettings
<<***extension***>> *(p. 213)* Configures the distribution of telemetry data . . . 1195
DDS_MonitoringEventDistributionSettings
<<***extension***>> *(p. 213)* Configures the distribution of event metrics . . . 1197
DDS_MonitoringLoggingDistributionSettings
<<***extension***>> *(p. 213)* Configures the distribution of log messages . . . 1200
DDS_MonitoringLoggingForwardingSettings
<<***extension***>> *(p. 213)* Configures the forwarding levels of log messages for the different **NDDS_Config_LogFacility** (p. 727) . . . 1204
DDS_MonitoringMetricSelection
<<***extension***>> *(p. 213)* This data structure is used to configure event and periodic metrics collection and distribution for a specific set of observable resources . . . 1206
DDS_MonitoringMetricSelectionSeq
Declares IDL sequence < **DDS_MonitoringMetricSelection** (p. 1206) > . . . 1208
DDS_MonitoringPeriodicDistributionSettings
<<***extension***>> *(p. 213)* Configures the distribution of periodic metrics . . . 1209
DDS_MonitoringQosPolicy
<<***extension***>> *(p. 213)* Configures the use of RTI Monitoring Library 2.0 to collect and distribute RTI Connext telemetry data . . . 1210
DDS_MonitoringTelemetryData
<<***extension***>> *(p. 213)* Configures the telemetry data that will be distributed . . . 1213
DDS_MultiChannelQosPolicy
<<***extension***>> *(p. 213)* Configures the ability of a DataWriter to send data on different multicast groups (addresses) based on the value of the data . . . 1214
DDS_Octets
Built-in type consisting of a variable-length array of opaque bytes . . . 1217
DDS_OctetSeq
Instantiates **FooSeq** (p. 1965) < **DDS_Octet** (p. 514) > . . . 1218
DDS_OctetsSeq
Instantiates **FooSeq** (p. 1965) < **DDS_Octets** (p. 1217) > . . . 1219
DDS_OfferedDeadlineMissedStatus
DDS_OFFERED_DEADLINE_MISSED_STATUS (p. 539) . . . 1219
DDS_OfferedIncompatibleQosStatus
DDS_OFFERED_INCOMPATIBLE_QOS_STATUS (p. 540) . . . 1221
DDS_OwnershipQosPolicy
<<***spec***>> *(p. 209)* Specifies whether it is allowed for multiple **DDSDataWriter** (p. 1589) (s) to write the same instance of the data and if so, how these modifications should be arbitrated . . . 1222
DDS_OwnershipStrengthQosPolicy
Specifies the value of the strength used to arbitrate among multiple **DDSDataWriter** (p. 1589) objects that attempt to modify the same instance of a data type (identified by **DDSTopic** (p. 1885) + key) . . . 1227
DDS_ParticipantBuiltinTopicData
Entry created when a DomainParticipant object is discovered . . . 1229
DDS_ParticipantBuiltinTopicDataSeq
Instantiates **FooSeq** (p. 1965) < **DDS_ParticipantBuiltinTopicData** (p. 1229) > . . . 1235

DDS_ParticipantTrustAlgorithmInfo
Trust Plugins algorithm information associated with the discovered DomainParticipant 1235
DDS_ParticipantTrustInterceptorAlgorithmInfo
Trust Plugins interception algorithm information associated with the discovered DomainParticipant 1236
DDS_ParticipantTrustKeyEstablishmentAlgorithmInfo
Trust Plugins key establishment algorithm information associated with the discovered Domain↩ Participant 1237
DDS_ParticipantTrustProtectionInfo
Trust Plugins Protection information associated with the discovered DomainParticipant 1238
DDS_ParticipantTrustSignatureAlgorithmInfo
Trust Plugins signature algorithm information associated with the discovered DomainParticipant . . 1239
DDS_PartitionQosPolicy
<<***spec***>> *(*p. *209)* Set of strings that introduces logical partitions in **DDSDomainParticipant** (p. 1619), **DDSPublisher** (p. 1818), or **DDSSubscriber** (p. 1860) entities 1240
DDS_PersistentStorageSettings
Configures durable writer history and durable reader state 1242
DDS_PresentationQosPolicy
<<***spec***>> *(*p. *209)* Specifies how the samples representing changes to data instances are presented to a subscribing application 1249
DDS_PrintFormatProperty
A collection of attributes used to configure how data samples will be formatted when converted to a string 1253
DDS_ProductVersion_t
<<***extension***>> *(*p. *432)* Type used to represent the current version of RTI Connext 1255
DDS_ProfileQosPolicy
<<***extension***>> *(*p. *213)* Configures the way that XML documents containing QoS profiles are loaded by RTI Connext 1257
DDS_Property_t
Properties are name/value pairs objects 1259
DDS_PropertyQosPolicy
<<***extension***>> *(*p. *213)* Stores name/value(string) pairs that can be used to configure certain parameters of RTI Connext that are not exposed through formal QoS policies. Can also be used to store and propagate application-specific name/value pairs that can be retrieved by user code during discovery 1260
DDS_PropertySeq
Declares IDL `sequence` < **DDS_Property_t** (p. 1259) > 1262
DDS_ProtocolVersion_t
<<***extension***>> *(*p. *432)* Type used to represent the version of the RTPS protocol 1263
DDS_PublicationBuiltinTopicData
Entry created when a **DDSDataWriter** (p. 1589) is discovered in association with its Publisher . . . 1264
DDS_PublicationBuiltinTopicDataSeq
Instantiates **`FooSeq`** (p. 1965) < **DDS_PublicationBuiltinTopicData** (p. 1264) > 1273
DDS_PublicationMatchedStatus
DDS_PUBLICATION_MATCHED_STATUS (p. 543) 1273
DDS_PublisherQos
QoS policies supported by a **DDSPublisher** (p. 1818) entity 1276
DDS_PublishModeQosPolicy
<<***extension***>> *(*p. *213)* Specifies how RTI Connext sends application data on the network. This QoS policy can be used to tell RTI Connext to use its *own* thread to send data, instead of the user thread 1279
DDS_QosPolicyCount
Type to hold a counter for a **DDS_QosPolicyId_t** (p. 557) 1283
DDS_QosPolicyCountSeq
Declares IDL `sequence` < **DDS_QosPolicyCount** (p. 1283) > 1284

DDS_QosPrintAll_t
Special type which is used to select the to_string overloads when printing Qos objects 1285
DDS_QosPrintFormat
A collection of attributes used to configure how a QoS appears when printed 1285
DDS_QueryConditionParams
<<***extension***>> *(p. 432)* Input parameters for **DDSDataReader::create_querycondition_w_↩ params** (p. 1562) 1287
DDS_ReadConditionParams
<<***extension***>> *(p. 432)* Input parameters for **DDSDataReader::create_readcondition_w_↩ params** (p. 1561) 1288
DDS_ReaderDataLifecycleQosPolicy
<<***spec***>> *(p. 209)* Controls how a DataReader manages the lifecycle of the data that it has received 1289
DDS_ReceiverPoolQosPolicy
<<***extension***>> *(p. 213)* Configures threads used by RTI Connext to receive and process data from transports (for example, UDP sockets) 1293
DDS_ReliabilityQosPolicy
<<***extension***>> *(p. 213)* <<***spec***>> *(p. 209)* Indicates the level of reliability offered/requested by RTI Connext 1295
DDS_ReliableReaderActivityChangedStatus
<<***extension***>> *(p. 432)* Describes the activity (i.e. are acknowledgements forthcoming) of reliable readers matched to a reliable writer 1300
DDS_ReliableWriterCacheChangedStatus
<<***extension***>> *(p. 432)* A summary of the state of a data writer's cache of unacknowledged samples written 1302
DDS_ReliableWriterCacheEventCount
<<***extension***>> *(p. 432)* The number of times the number of unacknowledged samples in the cache of a reliable writer hit a certain well-defined threshold 1305
DDS_RequestedDeadlineMissedStatus
DDS_REQUESTED_DEADLINE_MISSED_STATUS (p. 540) 1306
DDS_RequestedIncompatibleQosStatus
DDS_REQUESTED_INCOMPATIBLE_QOS_STATUS (p. 540) 1307
DDS_ResourceLimitsQosPolicy
<<***spec***>> *(p. 209)* Controls the amount of physical memory allocated for DDS entities, if dynamic allocations are allowed, and how they occur. Also controls memory usage among different instance values for keyed topics 1308
DDS_RTPS_EntityId_t
From the DDS-RTPS specification: type used to hold the suffix part of the globally-unique RTPS-entity identifiers 1313
DDS_RTPS_GUID_t
From the DDS-RTPS specification: type used to hold a globally-unique RTPS-entity identifier . . . 1314
DDS_RtpsReliableReaderProtocol_t
<<***extension***>> *(p. 213)* Qos related to reliable reader protocol defined in RTPS 1314
DDS_RtpsReliableWriterProtocol_t
<<***extension***>> *(p. 213)* QoS related to the reliable writer protocol defined in RTPS 1320
DDS_RtpsWellKnownPorts_t
<<***extension***>> *(p. 213)* RTPS well-known port mapping configuration 1338
DDS_SampleIdentity_t
Type definition for a Sample Identity 1343
DDS_SampleInfo
Information that accompanies each sample that is `read` or `taken` 1344
DDS_SampleInfoSeq
Declares IDL `sequence` < **DDS_SampleInfo** (p. 1344) > 1355

DDS_SampleLostStatus
DDS_SAMPLE_LOST_STATUS (p. 541) . . . 1355
DDS_SampleRejectedStatus
DDS_SAMPLE_REJECTED_STATUS (p. 541) . . . 1356
DDS_SequenceNumber_t
Type for *sequence* number representation . . . 1357
DDS_ServiceQosPolicy
<<***extension***>> *(p. 213)* Service or tool associated with a DDS entity . . . 1358
DDS_ServiceRequest
A request coming from one of the built-in services . . . 1359
DDS_ServiceRequestAcceptedStatus
DDS_SERVICE_REQUEST_ACCEPTED_STATUS (p. 544) . . . 1361
DDS_ServiceRequestSeq
Instantiates **FooSeq** (p. 1965) < **DDS_ServiceRequest** (p. 1359) > . . . 1363
DDS_ShortSeq
Instantiates **FooSeq** (p. 1965) < **DDS_Short** (p. 514) > . . . 1363
DDS_StringSeq
Instantiates **FooSeq** (p. 1965) < char* > with value type semantics . . . 1364
DDS_StructMember
A description of a member of a struct . . . 1365
DDS_StructMemberSeq
Defines a sequence of struct members . . . 1367
DDS_SubscriberQos
QoS policies supported by a **DDSSubscriber** (p. 1860) entity . . . 1367
DDS_SubscriptionBuiltinTopicData
Entry created when a **DDSDataReader** (p. 1556) is discovered in association with its Subscriber . . . 1370
DDS_SubscriptionBuiltinTopicDataSeq
Instantiates **FooSeq** (p. 1965) < **DDS_SubscriptionBuiltinTopicData** (p. 1370) > . . . 1379
DDS_SubscriptionMatchedStatus
DDS_SUBSCRIPTION_MATCHED_STATUS (p. 543) . . . 1379
DDS_SystemResourceLimitsQosPolicy
<<***extension***>> *(p. 213)* Configures **DDSDomainParticipant** (p. 1619)-independent resources used by RTI Connext. Mainly used to change the maximum number of **DDSDomainParticipant** (p. 1619) entities that can be created within a single process (address space) . . . 1382
DDS_Tag
Tags are name/value pair objects . . . 1384
DDS_TagSeq
Declares IDL sequence < **DDS_Tag** (p. 1384) > . . . 1385
DDS_ThreadSettings_t
The properties of a thread of execution . . . 1385
DDS_Time_t
Type for *time* representation . . . 1387
DDS_TimeBasedFilterQosPolicy
<<***spec***>> *(p. 209)* Filter that allows a **DDSDataReader** (p. 1556) to specify that it is interested only in (potentially) a subset of the values of the data . . . 1389
DDS_TopicBuiltinTopicData
Entry created when a Topic object discovered . . . 1391
DDS_TopicBuiltinTopicDataSeq
Instantiates **FooSeq** (p. 1965) < **DDS_TopicBuiltinTopicData** (p. 1391) > . . . 1396
DDS_TopicDataQosPolicy
Attaches a buffer of opaque data that is distributed by means of **Built-in Topics** (p. 61) during discovery . . . 1396
DDS_TopicQos
QoS policies supported by a **DDSTopic** (p. 1885) entity . . . 1397

DDS_TopicQueryData
<<***extension***>> *(p. 432)* Provides information about a **DDSTopicQuery** (p. 1895) 1402
DDS_TopicQueryDispatchQosPolicy
<<***extension***>> *(p. 213)* Configures the ability of a **DDSDataWriter** (p. 1589) to publish samples in response to a **DDSTopicQuery** (p. 1895) 1404
DDS_TopicQuerySelection
<<***extension***>> *(p. 432)* Specifies the data query that defines a **DDSTopicQuery** (p. 1895) . . . 1406
DDS_TransportBuiltinQosPolicy
<<***extension***>> *(p. 213)* Specifies which built-in transports are used 1408
DDS_TransportInfo_t
Contains the class_id and message_size_max of an installed transport 1409
DDS_TransportInfoSeq
Instantiates **FooSeq** (p. 1965) < **DDS_TransportInfo_t** (p. 1409) > 1410
DDS_TransportMulticastMapping_t
<<***extension***>> *(p. 213)* Type representing a list of multicast mapping elements 1410
DDS_TransportMulticastMappingFunction_t
Type representing an external mapping function 1412
DDS_TransportMulticastMappingQosPolicy
<<***extension***>> *(p. 213)* Specifies a list of topic_expressions and multicast addresses that can be used by an Entity with a specific topic name to receive data 1413
DDS_TransportMulticastMappingSeq
Declares IDL sequence< **DDS_TransportMulticastMapping_t** (p. 1410) > 1415
DDS_TransportMulticastQosPolicy
<<***extension***>> *(p. 213)* Specifies the multicast address on which a **DDSDataReader** (p. 1556) wants to receive its data. It can also specify a port number as well as a subset of the available (at the **DDSDomainParticipant** (p. 1619) level) transports with which to receive the multicast data 1415
DDS_TransportMulticastSettings_t
Type representing a list of multicast locators 1417
DDS_TransportMulticastSettingsSeq
Declares IDL sequence< **DDS_TransportMulticastSettings_t** (p. 1417) > 1419
DDS_TransportPriorityQosPolicy
<<***spec***>> *(p. 209)* This QoS policy allows the application to take advantage of transports that are capable of sending messages with different priorities 1419
DDS_TransportSelectionQosPolicy
<<***extension***>> *(p. 213)* Specifies the physical transports a **DDSDataWriter** (p. 1589) or **DDSDataReader** (p. 1556) may use to send or receive data 1421
DDS_TransportUnicastQosPolicy
<<***extension***>> *(p. 213)* Specifies a subset of transports and a port number that can be used by an Entity to receive data 1422
DDS_TransportUnicastSettings_t
Type representing a list of unicast locators 1424
DDS_TransportUnicastSettingsSeq
Declares IDL sequence< **DDS_TransportUnicastSettings_t** (p. 1424) > 1426
DDS_TrustAlgorithmRequirements
Type to describe Trust Plugins algorithm requirements for an entity 1426
DDS_TypeAllocationParams_t
Configures whether or not to allocate pointer and optional members 1427
DDS_TypeCode
The definition of a particular data type, which you can use to inspect the name, members, and other properties of types generated with rtiddsgen (see the Code Generator User's Manual) or to modify types you define yourself at runtime 1428
DDS_TypeCodeFactory
A singleton factory for creating, copying, and deleting data type definitions dynamically 1476

DDS_TypeCodePrintFormatProperty
A collection of attributes used to configure how a TypeCode appears when converted to a string . . 1490
DDS_TypeConsistencyEnforcementQosPolicy
<<***spec***>> *(p. 209)* Defines the rules for determining whether the type used to publish a given topic is consistent with that used to subscribe to it . . . 1493
DDS_TypeDeallocationParams_t
Configures whether to release or not pointer and optional members . . . 1497
DDS_TypeSupportQosPolicy
[DEPRECATED] (p. 384) <<***extension***>> *(p. 213)* Allows you to attach application-specific values to a **DDSDataWriter** (p. 1589) or **DDSDataReader** (p. 1556), which are passed to the serialization or deserialization routine of the associated data type and choose whether to set padding bytes to zero during serialization . . . 1499
DDS_UInt8Seq
Instantiates **FooSeq** (p. 1965) < **DDS_UInt8** (p. 514) > . . . 1500
DDS_UnionMember
A description of a member of a union . . . 1501
DDS_UnionMemberSeq
Defines a sequence of union members . . . 1502
DDS_UnsignedLongLongSeq
Instantiates **FooSeq** (p. 1965) < **DDS_UnsignedLongLong** (p. 515) > . . . 1503
DDS_UnsignedLongSeq
Instantiates **FooSeq** (p. 1965) < **DDS_UnsignedLong** (p. 515) > . . . 1503
DDS_UnsignedShortSeq
Instantiates **FooSeq** (p. 1965) < **DDS_UnsignedShort** (p. 514) > . . . 1503
DDS_UserDataQosPolicy
Attaches a buffer of opaque data that is distributed by means of **Built-in Topics** (p. 61) during discovery . . . 1504
DDS_ValueMember
A description of a member of a value type . . . 1505
DDS_ValueMemberSeq
Defines a sequence of value members . . . 1508
DDS_VendorId_t
<<***extension***>> *(p. 432)* Type used to represent the vendor of the service implementing the RTPS protocol . . . 1508
DDS_VirtualWriterInfo
The structure that contains the information of a virtual writer . . . 1509
DDS_WaitSetProperty_t
<<***extension***>> *(p. 432)* Specifies the **DDSWaitSet** (p. 1897) behavior for multiple trigger events 1509
DDS_WcharSeq
Instantiates **FooSeq** (p. 1965) < **DDS_Wchar** (p. 514) > . . . 1510
DDS_WireProtocolQosPolicy
<<***extension***>> *(p. 213)* Specifies the wire-protocol-related attributes for the **DDSDomainParticipant** (p. 1619) . . . 1511
DDS_WriteParams_t
<<***extension***>> *(p. 432)* Input parameters for writing with **FooDataWriter::write_w_params** (p. 1956), **FooDataWriter::dispose_w_params** (p. 1959), **FooDataWriter::register_instance_w_params** (p. 1948), **FooDataWriter::unregister_instance_w_params** (p. 1951) . . . 1519
DDS_WriterDataLifecycleQosPolicy
<<***spec***>> *(p. 209)* Controls how a **DDSDataWriter** (p. 1589) handles the lifecycle of the instances (keys) that it is registered to manage . . . 1524
DDS_WstringSeq
Instantiates **FooSeq** (p. 1965) < **DDS_Wchar** (p. 514)* > . . . 1527

DDSAsyncWaitSet
A class for dispatching **DDSCondition** (p. 1544) objects using separate threads of execution. You can see this class as an extension of a **DDSWaitSet** (p. 1897) that allows asynchronously waiting for the attached **DDSCondition** (p. 1544) objects to trigger and provide a notification by calling **DDSCondition::dispatch** (p. 1546) 1528
DDSAsyncWaitSetCompletionToken
<<***interface***>> *(p. 433)* Implementation of the completion token role element of the asynchronous completion token pattern that is part of the **DDSAsyncWaitSet** (p. 1528) behavior 1541
DDSAsyncWaitSetListener
<<***interface***>> *(p. 433)* Listener for receiving event notifications related to the thread pool of the **DDSAsyncWaitSet** (p. 1528) 1543
DDSCondition
<<***interface***>> *(p. 433)* Root class for all the conditions that may be attached to a **DDSWaitSet** (p. 1897) 1544
DDSConditionHandler
<<***extension***>> *(p. 432)* <<***interface***>> *(p. 433)* Handler called by the **DDSCondition::dispatch** (p. 1546) 1546
DDSConditionSeq
Instantiates **FooSeq** (p. 1965) < **DDSCondition** (p. 1544) > 1547
DDSContentFilter
<<***interface***>> *(p. 433)* Interface to be used by a custom filter of a **DDSContentFilteredTopic** (p. 1551) 1548
DDSContentFilteredTopic
<<***interface***>> *(p. 433)* Specialization of **DDSTopicDescription** (p. 1892) that allows for content-based subscriptions 1551
DDSDataReader
<<***interface***>> *(p. 433)* Allows the application to: (1) declare the data it wishes to receive (i.e. make a subscription) and (2) access the data received by the attached **DDSSubscriber** (p. 1860) . 1556
DDSDataReaderListener
<<***interface***>> *(p. 433)* **DDSListener** (p. 1794) for reader status 1583
DDSDataReaderSeq
Declares IDL sequence < **DDSDataReader** (p. 1556) > 1585
DDSDataReaderStatusConditionHandler
<<***interface***>> *(p. 433)* Realization of a **DDSConditionHandler** (p. 1546) that handles the status of a **DDSDataReader** (p. 1556) 1586
DDSDataTagQosPolicyHelper
Policy helpers that facilitate management of the data tags in the input policy 1588
DDSDataWriter
<<***interface***>> *(p. 433)* Allows an application to set the value of the data to be published under a given **DDSTopic** (p. 1885) 1589
DDSDataWriterListener
<<***interface***>> *(p. 433)* **DDSListener** (p. 1794) for writer status 1612
DDSDomainEntity
<<***interface***>> *(p. 433)* Abstract base class for all DDS entities except for the **DDSDomainParticipant** (p. 1619) 1618
DDSDomainParticipant
<<***interface***>> *(p. 433)* Container for all **DDSDomainEntity** (p. 1618) objects 1619
DDSDomainParticipantFactory
<<***singleton***>> *(p. 433)* <<***interface***>> *(p. 433)* Allows creation and destruction of **DDSDomainParticipant** (p. 1619) objects 1694
DDSDomainParticipantListener
<<***interface***>> *(p. 433)* Listener for participant status 1722
DDSDynamicDataReader
Reads (subscribes to) objects of type **DDS_DynamicData** (p. 1013) 1724

DDSDynamicDataTypeSupport
A factory for registering a dynamically defined type and creating **DDS_DynamicData** (p. 1013) objects . . . 1725
DDSDynamicDataWriter
Writes (publishes) objects of type **DDS_DynamicData** (p. 1013) . . . 1730
DDSEntity
<<***interface***>> *(p. 433)* Abstract base class for all the DDS objects that support QoS policies, a listener, and a status condition . . . 1731
DDSFlowController
<<***interface***>> *(p. 433)* A flow controller is the object responsible for shaping the network traffic by determining when attached asynchronous **DDSDataWriter** (p. 1589) instances are allowed to write data . . . 1736
DDSGuardCondition
<<***interface***>> *(p. 433)* A specific **DDSCondition** (p. 1544) whose `trigger_value` is completely under the control of the application . . . 1739
DDSKeyedOctetsDataReader
<<***interface***>> *(p. 433)* Instantiates `DataReader` < **DDS_KeyedOctets** (p. 1168) > . . . 1741
DDSKeyedOctetsDataWriter
<<***interface***>> *(p. 433)* Instantiates `DataWriter` < **DDS_KeyedOctets** (p. 1168) > . . . 1750
DDSKeyedOctetsTypeSupport
<<***interface***>> *(p. 433)* **DDS_KeyedOctets** (p. 1168) type support . . . 1763
DDSKeyedStringDataReader
<<***interface***>> *(p. 433)* Instantiates `DataReader` < **DDS_KeyedString** (p. 1171) > . . . 1769
DDSKeyedStringDataWriter
<<***interface***>> *(p. 433)* Instantiates `DataWriter` < **DDS_KeyedString** (p. 1171) > . . . 1778
DDSKeyedStringTypeSupport
<<***interface***>> *(p. 433)* Keyed string type support . . . 1788
DDSListener
<<***interface***>> *(p. 433)* Abstract base class for all Listener interfaces . . . 1794
DDSMultiTopic
[Not supported (optional)] <<***interface***>> *(p. 433)* A specialization of **DDSTopicDescription** (p. 1892) that allows subscriptions that combine/filter/rearrange data coming from several topics . . . 1797
DDSOctetsDataReader
<<***interface***>> *(p. 433)* Instantiates `DataReader` < **DDS_Octets** (p. 1217) > . . . 1800
DDSOctetsDataWriter
<<***interface***>> *(p. 433)* Instantiates `DataWriter` < **DDS_Octets** (p. 1217) > . . . 1804
DDSOctetsTypeSupport
<<***interface***>> *(p. 433)* **DDS_Octets** (p. 1217) type support . . . 1811
DDSParticipantBuiltinTopicDataDataReader
Instantiates `DataReader` < **DDS_ParticipantBuiltinTopicData** (p. 1229) > . . . 1816
DDSParticipantBuiltinTopicDataTypeSupport
Instantiates `TypeSupport` < **DDS_ParticipantBuiltinTopicData** (p. 1229) > . . . 1816
DDSPropertyQosPolicyHelper
Policy helpers that facilitate management of the properties in the input policy . . . 1817
DDSPublicationBuiltinTopicDataDataReader
Instantiates `DataReader` < **DDS_PublicationBuiltinTopicData** (p. 1264) > . . . 1817
DDSPublicationBuiltinTopicDataTypeSupport
Instantiates `TypeSupport` < **DDS_PublicationBuiltinTopicData** (p. 1264) > . . . 1818
DDSPublisher
<<***interface***>> *(p. 433)* A publisher is the object responsible for the actual dissemination of publications . . . 1818
DDSPublisherListener
<<***interface***>> *(p. 433)* **DDSListener** (p. 1794) for **DDSPublisher** (p. 1818) status . . . 1839

DDSPublisherSeq
Declares IDL `sequence` < **DDSPublisher** (p. 1818) > . . . 1840
DDSQueryCondition
<<***interface***>> *(p. 433)* These are specialised **DDSReadCondition** (p. 1842) objects that allow the application to also specify a filter on the locally available data . . . 1841
DDSReadCondition
<<***interface***>> *(p. 433)* Conditions specifically dedicated to read operations and attached to one **DDSDataReader** (p. 1556) . . . 1842
DDSServiceRequestDataReader
Instantiates `DataReader` < **DDS_ServiceRequest** (p. 1359) > . . . 1845
DDSServiceRequestTypeSupport
Instantiates `TypeSupport` < **DDS_ServiceRequest** (p. 1359) > . . . 1845
DDSStatusCondition
<<***interface***>> *(p. 433)* A specific **DDSCondition** (p. 1544) that is associated with each **DDSEntity** (p. 1731) . . . 1846
DDSStringDataReader
<<***interface***>> *(p. 433)* Instantiates `DataReader` < char* > . . . 1848
DDSStringDataWriter
<<***interface***>> *(p. 433)* Instantiates `DataWriter` < char* > . . . 1852
DDSStringTypeSupport
<<***interface***>> *(p. 433)* String type support . . . 1855
DDSSubscriber
<<***interface***>> *(p. 433)* A subscriber is the object responsible for actually receiving data from a subscription . . . 1860
DDSSubscriberListener
<<***interface***>> *(p. 433)* **DDSListener** (p. 1794) for status about a subscriber . . . 1881
DDSSubscriberSeq
Declares IDL `sequence` < **DDSSubscriber** (p. 1860) > . . . 1882
DDSSubscriptionBuiltinTopicDataDataReader
Instantiates `DataReader` < **DDS_SubscriptionBuiltinTopicData** (p. 1370) > . . . 1882
DDSSubscriptionBuiltinTopicDataTypeSupport
Instantiates `TypeSupport` < **DDS_SubscriptionBuiltinTopicData** (p. 1370) > . . . 1883
DDSThreadFactory
<<***extension***>> *(p. 432)* <<***interface***>> *(p. 433)* Interface for providing the threads needed by the middleware. It consists of operations to create and delete threads. The interface depends on the **DDSThreadFactory_OnSpawnedFunction** (p. 718) that specifies the operation to run in the new thread . . . 1883
DDSTopic
<<***interface***>> *(p. 433)* The most basic description of the data to be published and subscribed . . . 1885
DDSTopicBuiltinTopicDataDataReader
Instantiates `DataReader` < **DDS_TopicBuiltinTopicData** (p. 1391) > . . . 1890
DDSTopicBuiltinTopicDataTypeSupport
Instantiates `TypeSupport` < **DDS_TopicBuiltinTopicData** (p. 1391) > . . . 1891
DDSTopicDescription
<<***interface***>> *(p. 433)* Base class for **DDSTopic** (p. 1885), **DDSContentFilteredTopic** (p. 1551), and **DDSMultiTopic** (p. 1797) . . . 1892
DDSTopicListener
<<***interface***>> *(p. 433)* **DDSListener** (p. 1794) for **DDSTopic** (p. 1885) entities . . . 1894
DDSTopicQuery
<<***extension***>> *(p. 432)* Allows a **DDSDataReader** (p. 1556) to query the sample cache of its matching **DDSDataWriter** (p. 1589) . . . 1895
DDSTopicQueryHelper
Helpers to provide utility operations related to **DDSTopicQuery** (p. 1895) . . . 1896

DDSTypeSupport
<<***interface***>> *(p. 433)* An abstract *marker* interface that has to be specialized for each concrete user data type that will be used by the application . . . 1897
DDSWaitSet
<<***interface***>> *(p. 433)* Allows an application to wait until one or more of the attached **DDSCondition** (p. 1544) objects has a `trigger_value` of **DDS_BOOLEAN_TRUE** (p. 513) or else until the timeout expires . . . 1897
DDSWriterContentFilter
<<***interface***>> *(p. 433)* Interface to be used by a custom filter of a **DDSContentFilteredTopic** (p. 1551) . . . 1905
rti::flat::FinalAlignedArrayOffset< ElementOffset, N >
Offset to an array of final elements . . . 1909
rti::flat::FinalArrayOffset< ElementOffset, N >
Offset to an array of final elements . . . 1911
rti::flat::FinalOffset< T >
The base class of all Offsets to a final struct type . . . 1913
rti::flat::FinalSequenceBuilder< ElementOffset >
Builds a sequence member of fixed-size elements . . . 1913
rti::flat::flat_type_traits< T >
Given a **Sample** (p. 2187), an Offset or a Builder, it allows obtaining the other types . . . 1915
Foo
A representative user-defined data type . . . 1916
FooDataReader
<<***interface***>> *(p. 433)* <<***generic***>> *(p. 433)* User data type-specific data reader . . . 1917
FooDataWriter
<<***interface***>> *(p. 433)* <<***generic***>> *(p. 433)* User data type specific data writer . . . 1944
FooSeq
<<***interface***>> *(p. 433)* <<***generic***>> *(p. 433)* A type-safe, ordered collection of elements. The type of these elements is referred to in this documentation as **`Foo`** (p. 1916) . . . 1965
FooTypeSupport
<<***interface***>> *(p. 433)* <<***generic***>> *(p. 433)* User data type specific interface . . . 1979
connext::IllegalOperationException
The operation was called under improper circumstances . . . 1992
connext::ImmutablePolicyException
Application attempted to modify an immutable QoS policy . . . 1992
connext::InconsistentPolicyException
Application specified a set of QoS policies that are not consistent with each other . . . 1993
connext::IsInvalidSamplePredicate< T >
Predicate-class to determine if a sample contains invalid data . . . 1993
connext::IsReplyRelatedPredicate< T >
Predicate-class to match replies by their related request . . . 1993
connext::IsValidSamplePredicate< T >
Predicate-class to determine if a sample contains valid data . . . 1995
connext::LoanedSamples< T >
Provides access to a collection of middleware-loaned samples . . . 1995
connext::LoanedSamples< T >::LoanMemento
A simple value-type for the internal representation of the a **LoanedSamples** (p. 1995) object . . . 2004
connext::LogicException
Base class of all RTI Connext exceptions caused by a logic error . . . 2005
connext::MessagingLibraryVersion
The Connext Messaging Library version . . . 2005
connext::MessagingVersion
The Connext Messaging version . . . 2006

rti::flat::MutableArrayBuilder< ElementBuilder, N >
Builds an array member of variable-size elements 2008
rti::flat::MutableArrayOffset< ElementOffset, N >
Offset to an array of variable-size elements 2010
rti::flat::MutableOffset
The base class of all Offsets to a final struct type 2011
rti::flat::MutableSequenceBuilder< ElementBuilder >
Builds a sequence member of variable-size elements 2012
MyFlatFinalOffset
Represents the Offset to an arbitrary user-defined FlatData final IDL struct 2014
MyFlatMutableBuilder
Represents the Builder for an arbitrary user-defined mutable type 2018
MyFlatMutableOffset
Represents the Offset to an arbitrary user-defined FlatData mutable IDL struct 2025
MyFlatUnionBuilder
Represents the Builder for an arbitrary user-defined mutable IDL union 2032
MyFlatUnionOffset
Represents the Offset to an arbitrary user-defined FlatData mutable IDL union 2035
NDDS_Config_LibraryVersion_t
The version of a single library shipped as part of an RTI Connext distribution 2039
NDDS_Config_LogMessage
A log message's input parameter to a logging device, including the text and additional information . 2040
NDDS_Transport_Address_t
Addresses are stored individually as network-ordered bytes 2042
NDDS_Transport_Interface_t
Storage for the description of a network interface used by a Transport Plugin 2043
NDDS_Transport_Property_t
Base configuration structure that must be inherited by derived Transport Plugin classes 2044
NDDS_Transport_Shmem_Property_t
Subclass of **NDDS_Transport_Property_t** (p. 2044) allowing specification of parameters that are specific to the shared-memory transport 2052
NDDS_Transport_UDP_WAN_CommPortsMappingInfo
Type for storing UDP WAN communication ports 2055
NDDS_Transport_UDPv4_Property_t
Configurable IPv4/UDP Transport-Plugin properties 2056
NDDS_Transport_UDPv4_WAN_Property_t
Configurable IPv4/UDP WAN Transport-Plugin properties 2067
NDDS_Transport_UDPv6_Property_t
Configurable IPv6/UDP Transport-Plugin properties 2077
NDDS_Transport_UUID
Unequivocally identifies a transport plugin instance 2087
NDDS_Utility_NetworkCaptureParams_t
Input parameters for starting network capture 2087
NDDSConfigActivityContext
Activity Context APIs 2089
NDDSConfigCompliance
APIs to configure all the compliance mask 2090
NDDSConfigLogger
<<interface>> (p. 433) The singleton type used to configure RTI Connext logging 2093
NDDSConfigLoggerDevice
<<interface>> (p. 433) Logging device interface. Use for user-defined logging devices 2100
NDDSConfigVersion
<<interface>> (p. 433) The version of an RTI Connext distribution 2101

NDDSLM
Other Utilities APIs . . . 2103
NDDSTransportSupport
<<***interface***>> *(p. 433)* The utility class used to configure RTI Connext pluggable transports . . . 2104
NDDSUtility
Other Utilities APIs . . . 2110
NDDSUtilityHeapMonitoring
Heap Monitoring APIs . . . 2113
NDDSUtilityNetworkCapture
Network Capture APIs . . . 2117
connext::NotEnabledException
Operation invoked on a **DDSEntity** (p. 1731) that is not yet enabled . . . 2126
rti::flat::OffsetBase
Base class of all Offset types . . . 2126
connext::OutOfResourcesException
RTI Connext ran out of the resources needed to complete the operation . . . 2131
connext::PreconditionNotMetException
A pre-condition for the operation was not met . . . 2132
rti::flat::PrimitiveArrayOffset< T, N >
Offset to an array of primitive elements . . . 2132
rti::flat::PrimitiveConstOffset< T >
A const Offset to an optional primitive member . . . 2133
rti::flat::PrimitiveOffset< T >
An Offset to an optional primitive member . . . 2134
rti::flat::PrimitiveSequenceBuilder< T >
Builds a sequence of primitive members . . . 2135
rti::flat::PrimitiveSequenceOffset< T >
Offset to a sequence of primitive elements . . . 2138
connext::Replier< TReq, TRep >
Allows receiving requests and sending replies . . . 2139
connext::ReplierListener< TReq, TRep >
Called when a **connext::Replier** (p. 2139) has new available requests . . . 2151
connext::ReplierParams< TReq, TRep >
Contains the parameters for creating a **connext::Replier** (p. 2139) . . . 2152
connext::Requester< TReq, TRep >
Allows sending requests and receiving replies . . . 2157
connext::RequesterParams
Contains the parameters for creating a **connext::Requester** (p. 2157) . . . 2178
connext::RuntimeException
Generic, unspecified error . . . 2183
connext::Sample< T >
A data sample and related info received from the middleware . . . 2183
rti::flat::Sample< OffsetType >
The generic definition of FlatData topic-types . . . 2187
connext::SampleIterator< T, IsConst >
STL-compliant random-access iterator for SampleRef<T> . . . 2191
connext::SampleRef< T >
A data sample and related information received from the middleware . . . 2191
rti::flat::SequenceIterator< E, OffsetKind >
Iterator for collections of Offsets . . . 2198
rti::flat::SequenceOffset< ElementOffset >
Offset to a sequence of non-primitive elements . . . 2204
connext::SimpleReplier< TReq, TRep >
A callback-based replier . . . 2206

connext::SimpleReplierListener< TReq, TRep >
The listener called by a **SimpleReplier** (p. 2206) . . . 2208
connext::SimpleReplierParams< TReq, TRep >
Contains the parameters for creating a **connext::SimpleReplier** (p. 2206) . . . 2209
rti::flat::StringBuilder
Builds a string . . . 2214
rti::flat::StringOffset
Offset to a string . . . 2216
connext::TimeoutException
The operation timed out (does not apply to wait or receive operations) . . . 2217
TransportAllocationSettings_t
Allocation settings used by various internal buffers . . . 2218
rti::flat::UnionBuilder< Discriminator >
Base class of builders for user-defined mutable unions . . . 2218
connext::UnsupportedException
Unsupported operation . . . 2219
connext::WriteSample< T >
A sample for writing data . . . 2219
connext::WriteSampleRef< T >
A reference to a data sample for writing . . . 2223

Chapter 6

File Index

6.1 File List

Here is a list of all documented files with brief descriptions:

howto_prototypes.h . . . 2227
transport_common.h . . . 2228
transport_common_user.h . . . 2233
transport_interface_user.h . . . 2242
transport_shmem.h . . . 2245
transport_udp.h . . . 2249
transport_udpv4.h . . . 2262
transport_udpv6.h . . . 2273
AggregationBuilders.hpp . . . 2281
Builder.hpp . . . 2287
BuilderHelper.hpp . . . 2295
ExceptionHelper.hpp . . . 2298
FlatSample.hpp . . . 2300
FlatSampleImpl.hpp . . . 2303
FlatTypeTraits.hpp . . . 2304
Offset.hpp . . . 2305
rtiflat.hpp . . . 2314
SequenceBuilders.hpp . . . 2315
SequenceIterator.hpp . . . 2324
SequenceOffsets.hpp . . . 2327
Interpreter.hpp . . . 2334
Stream.hpp . . . 2338

Chapter 7

Module Documentation

7.1 RTI Connext Exceptions

RTI Connext return-code exceptions.

Classes

- class **connext::RuntimeException**

  Generic, unspecified error.
- class **connext::LogicException**

  Base class of all RTI Connext exceptions caused by a logic error.
- class **connext::UnsupportedException**

  Unsupported operation.
- class **connext::BadParameterException**

  Illegal parameter value.
- class **connext::PreconditionNotMetException**

  A pre-condition for the operation was not met.
- class **connext::ImmutablePolicyException**

  Application attempted to modify an immutable QoS policy.
- class **connext::InconsistentPolicyException**

  Application specified a set of QoS policies that are not consistent with each other.
- class **connext::NotEnabledException**

  *Operation invoked on a **DDSEntity** (p. 1731) that is not yet enabled.*
- class **connext::AlreadyDeletedException**

  The object target of this operation has already been deleted.
- class **connext::IllegalOperationException**

  The operation was called under improper circumstances.
- class **connext::TimeoutException**

  The operation timed out (does not apply to wait or receive operations)
- class **connext::OutOfResourcesException**

  RTI Connext ran out of the resources needed to complete the operation.

7.1.1 Detailed Description

RTI Connext return-code exceptions.

The request-reply API reports internal DDS errors as exceptions. Each RTI Connext return code maps to a exception.

All exceptions inherit from the C++ standard exceptions `std::logic_error` or `std::runtime_error`.

Note that RETCODE_TIMEOUT and RETCODE_NO_DATA are not thrown in the request-reply API, because they are not considered errors. For example in case of no data or timeouts operations may return empty containers (e.g. **connext::Requester::take_replies(int)** (p. 2170)) or false (e.g. **connext::Requester::wait_for_replies(int, const Duration_t&)** (p. 2167)).

See also

Return Codes (p. 531)

Error handling example (p. 426)

7.2 Clock Selection

APIs related to clock selection.

APIs related to clock selection.

RTI Connext uses clocks to measure time and generate timestamps.

RTI Connext uses three clocks, an "internal clock," an "external clock," and a "wall clock". The internal clock measures time internally. The external clock generates timestamps, such as the source timestamp and the reception timestamp, in addition to providing the current time given by the Domain Participant's API. The internal and external clocks can be configured via properties. The wall clock is not configurable and always shows the system time. It is used internally whenever a real-world time is necessary, such as for external document time verification.

7.2.1 Available Clocks

Two clock implementations are generally available, the monotonic clock and the realtime clock.

The monotonic clock provides times that are monotonic from a clock that is not adjustable. This clock is useful to use in order to not be subject to changes in the system or realtime clock, which may be adjusted by the user or via time synchronization protocols. However, this time generally starts from an arbitrary point in time, such as system startup. Note that this clock is not available for all architectures. Please see the `Platform Notes` for the architectures on which it is supported. For the purposes of clock selection, this clock can be referenced by the name "monotonic".

The realtime clock provides the real time of the system. This clock may generally be monotonic but may not be guaranteed to be so. It is adjustable and may be subject to small and large changes in time. The time obtained from this clock is generally a meaningful time in that it is the amount of time from a known epoch. For the purposes of clock selection, this clock can be referenced by the names "realtime" or "system".

7.2.2 Clock Selection Strategy

By default, the internal clock uses the monotonic clock and the external clocks uses the realtime clock. The default setup offers robustness to changes in the system time. Since the monotonic clock is not available on all platforms, by default, RTI Connext will attempt to use the monotonic clock, and will fall back to the realtime clock if the monotonic clock is not available.

If you want your application to be robust to changes in the system time, you are not relying on source timestamps, and you want to avoid obtaining times from both clocks, you may use the monotonic clock for both the internal and external clocks.

7.2.3 Configuring Clock Selection

To configure the clock selection, use the **PROPERTY** (p. 617) QoS policy associated with the **DDSDomainParticipant** (p. 1619).

See also

DDS_PropertyQosPolicy (p. 1260)

The following table lists the supported clock selection properties.

Table 7.1 ***Clock Selection Properties***

Property	Description
dds.clock.external_clock	Comma-delimited list of clocks to use for the external clock, in order of preference. Valid clock names are "realtime", "system", and "monotonic". Default: "realtime"
dds.clock.internal_clock	Comma-delimited list of clocks to use for the internal clock, in order of preference. Valid clock names are "realtime", "system", and "monotonic". Default: "monotonic" if available, "realtime" if not

7.3 Domain Module

Contains the **DDSDomainParticipant** (p. 1619) class that acts as an entrypoint of RTI Connext and acts as a factory for many of the classes. The **DDSDomainParticipant** (p. 1619) also acts as a container for the other objects that make up RTI Connext.

Modules

- **DomainParticipantFactory**

  DDSDomainParticipantFactory *(p. 1694) entity and associated elements*
- **DomainParticipants**

  DDSDomainParticipant *(p. 1619) entity and associated elements*
- **Built-in Topics**

  Built-in objects created by RTI Connext but accessible to the application.

7.3.1 Detailed Description

Contains the **DDSDomainParticipant** (p. 1619) class that acts as an entrypoint of RTI Connext and acts as a factory for many of the classes. The **DDSDomainParticipant** (p. 1619) also acts as a container for the other objects that make up RTI Connext.

7.4 DomainParticipantFactory

DDSDomainParticipantFactory (p. 1694) entity and associated elements

Modules

- **DomainParticipantConfigParams**

  *<<**extension**>> (p. 432)* ***DDS_DomainParticipantConfigParams_t*** *(p. 964)*

Classes

- struct **DDS_DomainParticipantFactoryQos**

  QoS policies supported by a ***DDSDomainParticipantFactory*** *(p. 1694).*
- class **DDSDomainParticipantFactory**

  *<<**singleton**>> (p. 433) <<**interface**>> (p. 433) Allows creation and destruction of* ***DDSDomainParticipant*** *(p. 1619) objects.*

Macros

- #define **DDSTheParticipantFactory** **DDSDomainParticipantFactory::get_instance**()

  Can be used as an alias for the singleton factory returned by the operation ***DDSDomainParticipantFactory::get_instance*** *(p. 1697).*

Typedefs

- typedef **DDS_ReturnCode_t**(∗ **DDSDomainParticipantFactory_RegisterTypeFunction**) (**DDSDomainParticipant** ∗participant, const char ∗type_name)

  Prototype of a register type function.

Functions

- **DDS_Boolean DDS_DomainParticipantFactoryQos_equals** (const struct **DDS_DomainParticipantFactoryQos** *self, const struct **DDS_DomainParticipantFactoryQos** *other)

  *Compares two **DDS_DomainParticipantFactoryQos** (p. 967) for equality.*
- **DDS_ReturnCode_t DDS_DomainParticipantFactoryQos::print** () const

  *Prints this **DDS_DomainParticipantFactoryQos** (p. 967) to stdout.*
- **DDS_ReturnCode_t DDS_DomainParticipantFactoryQos::to_string** (char *string, **DDS_UnsignedLong** &string_size) const

  *Obtains a string representation of this **DDS_DomainParticipantQos** (p. 971).*
- **DDS_ReturnCode_t DDS_DomainParticipantFactoryQos::to_string** (char *string, **DDS_UnsignedLong** &string_size, const **DDS_DomainParticipantFactoryQos** &base) const

  *Obtains a string representation of this **DDS_DomainParticipantFactoryQos** (p. 967).*
- **DDS_ReturnCode_t DDS_DomainParticipantFactoryQos::to_string** (char *string, **DDS_UnsignedLong** &string_size, const **DDS_QosPrintFormat** &format) const

  *Obtains a string representation of this **DDS_DomainParticipantFactoryQos** (p. 967).*
- **DDS_ReturnCode_t DDS_DomainParticipantFactoryQos::to_string** (char *string, **DDS_UnsignedLong** &string_size, const **DDS_DomainParticipantFactoryQos** &base, const **DDS_QosPrintFormat** &format) const

  *Obtains a string representation of this **DDS_DomainParticipantFactoryQos** (p. 967).*
- **DDS_ReturnCode_t DDS_DomainParticipantFactoryQos::to_string** (char *string, **DDS_UnsignedLong** &string_size, const **DDS_QosPrintAll_t** &) const

  *Obtains a string representation of this **DDS_DomainParticipantFactoryQos** (p. 967).*
- **DDS_ReturnCode_t DDS_DomainParticipantFactoryQos::to_string** (char *string, **DDS_UnsignedLong** &string_size, const **DDS_QosPrintAll_t** &, const **DDS_QosPrintFormat** &format) const

  *Obtains a string representation of this **DDS_DomainParticipantFactoryQos** (p. 967).*

Variables

- const struct **DDS_DomainParticipantQos DDS_PARTICIPANT_QOS_DEFAULT**

  Special value for creating a DomainParticipant with default QoS.
- const struct **DDS_DomainParticipantConfigParams_t DDS_PARTICIPANT_CONFIG_PARAMS_DEFAULT**

  *Special value for creating a **DDSDomainParticipant** (p. 1619) from configuration using default parameters.*

7.4.1 Detailed Description

DDSDomainParticipantFactory (p. 1694) entity and associated elements

7.4.2 Macro Definition Documentation

7.4.2.1 DDSTheParticipantFactory

```
#define DDSTheParticipantFactory  DDSDomainParticipantFactory::get_instance()
```

Can be used as an alias for the singleton factory returned by the operation **DDSDomainParticipantFactory::get_instance** (p. 1697).

See also

DDSDomainParticipantFactory::get_instance (p. 1697)

Examples

HelloWorld_publisher.cxx, and **HelloWorld_subscriber.cxx**.

7.4.3 Typedef Documentation

7.4.3.1 DDSDomainParticipantFactory_RegisterTypeFunction

```
typedef  DDS_ReturnCode_t(* DDSDomainParticipantFactory_RegisterTypeFunction) ( DDSDomainParticipant
*participant, const char *type_name)
```

Prototype of a register type function.

Parameters

participant	<<***inout***>> *(p. 433)* **DDSDomainParticipant** (p. 1619) participant the type is registered with.
type_name	<<***in***>> *(p. 433)* Name the type is registered with.

Exceptions

One	of the **Standard Return Codes** (p. 532)

7.4.4 Function Documentation

7.4.4.1 DDS_DomainParticipantFactoryQos_equals()

```
DDS_Boolean DDS_DomainParticipantFactoryQos_equals (
            const struct  DDS_DomainParticipantFactoryQos * self,
            const struct  DDS_DomainParticipantFactoryQos * other )
```

Compares two **DDS_DomainParticipantFactoryQos** (p. 967) for equality.

Parameters

self	<<***in***>> *(p. 433)* This DomainParticipantFactoryQos.
other	<<***in***>> *(p. 433)* The other DomainParticipantFactoryQos to be compared with this DomainParticipantFactoryQos.

Returns

DDS_BOOLEAN_TRUE (p. 513) if the two Qos objects are equal or **DDS_BOOLEAN_FALSE** (p. 513) otherwise.

Referenced by **DDS_DomainParticipantFactoryQos::operator!=()**, and **DDS_DomainParticipantFactoryQos↩ ::operator==()**.

7.4.4.2 print()

```
DDS_ReturnCode_t DDS_DomainParticipantFactoryQos::print ( ) const  [inline]
```

Prints this **DDS_DomainParticipantFactoryQos** (p. 967) to stdout.

Only the differences between this **DDS_DomainParticipantFactoryQos** (p. 967) and the documented default are printed. If you wish to print everything regardless, see **DDS_DomainParticipantFactoryQos::to_string(char∗, DDS↩ _UnsignedLong&, const DDS_DomainParticipantFactoryQos&, const DDS_QosPrintFormat&) const** (p. 47). The output is formatted according to the default values for **DDS_QosPrintFormat** (p. 1285).

Returns

DDS_RETCODE_OK (p. 532) if no error was encountered.

7.4.4.3 to_string() [1/6]

```
DDS_ReturnCode_t DDS_DomainParticipantFactoryQos::to_string (
            char * string,
             DDS_UnsignedLong & string_size ) const  [inline]
```

Obtains a string representation of this **DDS_DomainParticipantQos** (p. 971).

Only the differences between this **DDS_DomainParticipantFactoryQos** (p. 967) and the documented default are printed to the string. If you wish to print everything regardless, see **DDS_DomainParticipantFactoryQos::to↩ _string(char∗, DDS_UnsignedLong&, const DDS_DomainParticipantFactoryQos&, const DDS_QosPrint↩ Format&) const** (p. 47). The output is formatted according to the default values for **DDS_QosPrintFormat** (p. 1285).

If the supplied buffer is NULL, the required length of the string is returned via the string_size parameter.

If the supplied buffer is not NULL, the string representation of the **DDS_DomainParticipantFactoryQos** (p. 967) is written to the buffer.

Parameters

string	<<***out***>> *(p. 433)* The char buffer that will be used to store the string representation of this **DDS_DomainParticipantFactoryQos** (p. 967). If NULL, this function will return the required length of this buffer through the string_size parameter.
string_size	<<***inout***>> *(p. 433)* Cannot be NULL. The size of the supplied buffer. If the supplied buffer is NULL, or if the supplied buffer is not large enough, this value will be updated with the required length of the buffer.

Returns

DDS_RETCODE_OK (p. 532) if no error was encountered.

See also

DDS_DomainParticipantFactoryQos::to_string(char*, DDS_UnsignedLong&, const DDS_DomainParticipantFactoryQos&, const DDS_QosPrintFormat&) const (p. 47)

7.4.4.4 to_string() [2/6]

```
DDS_ReturnCode_t DDS_DomainParticipantFactoryQos::to_string (
            char * string,
             DDS_UnsignedLong & string_size,
            const  DDS_DomainParticipantFactoryQos & base ) const  [inline]
```

Obtains a string representation of this **DDS_DomainParticipantFactoryQos** (p. 967).

This overload behaves the same as **DDS_DomainParticipantFactoryQos::to_string** (p. 45) but allows the caller to specify the **DDS_DomainParticipantFactoryQos** (p. 967), which is used as the base profile. Only the differences between the QoS and the base profile are included in the output string.

Parameters

string	<<***out***>> *(p. 433)* The char buffer that will be used to store the string representation of this **DDS_DomainParticipantFactoryQos** (p. 967). If NULL, this function will return the required length of this buffer through the string_size parameter.
string_size	<<***inout***>> *(p. 433)* Cannot be NULL. The size of the supplied buffer. If the supplied buffer is NULL, or if the supplied buffer is not large enough, this value will be updated with the required length of the buffer.
base	<<***in***>> *(p. 433)* The **DDS_DomainParticipantFactoryQos** (p. 967) to be used as the base QoS profile. Only the differences with respect to this base profile will be included in the output string.

Returns

DDS_RETCODE_OK (p. 532) if no error was encountered.

See also

DDS_DomainParticipantFactoryQos::to_string(char*, DDS_UnsignedLong&, const DDS_DomainParticipantFactoryQos&, const DDS_QosPrintFormat&) const (p. 47)

References **DDS_QosPrintFormat_INITIALIZER**.

7.4.4.5 to_string() [3/6]

```
DDS_ReturnCode_t DDS_DomainParticipantFactoryQos::to_string (
            char * string,
             DDS_UnsignedLong & string_size,
            const  DDS_QosPrintFormat & format ) const  [inline]
```

Obtains a string representation of this **DDS_DomainParticipantFactoryQos** (p. 967).

This overload behaves the same as **DDS_DomainParticipantFactoryQos::to_string** (p. 45) but allows the caller to specify the **DDS_QosPrintFormat** (p. 1285) which is used to format the output.

Parameters

string	<<***out***>> *(p. 433)* The char buffer that will be used to store the string representation of this **DDS_DomainParticipantFactoryQos** (p. 967). If NULL, this function will return the required length of this buffer through the string_size parameter.
string_size	<<***inout***>> *(p. 433)* Cannot be NULL. The size of the supplied buffer. If the supplied buffer is NULL, or if the supplied buffer is not large enough, this value will be updated with the required length of the buffer.
format	<<***in***>> *(p. 433)* The **DDS_QosPrintFormat** (p. 1285) to be used to format the output string.

Returns

DDS_RETCODE_OK (p. 532) if no error was encountered.

See also

DDS_DomainParticipantFactoryQos::to_string(char*, DDS_UnsignedLong&, const DDS_DomainParticipantFactoryQos&, const DDS_QosPrintFormat&) const (p. 47)

References **DDS_RETCODE_ERROR**, and **DDS_RETCODE_OK**.

7.4.4.6 to_string() [4/6]

```
DDS_ReturnCode_t DDS_DomainParticipantFactoryQos::to_string (
            char * string,
             DDS_UnsignedLong & string_size,
            const  DDS_DomainParticipantFactoryQos & base,
            const  DDS_QosPrintFormat & format ) const  [inline]
```

Obtains a string representation of this **DDS_DomainParticipantFactoryQos** (p. 967).

Only the differences between this **DDS_DomainParticipantFactoryQos** (p. 967) and the **DDS_DomainParticipant↩ FactoryQos** (p. 967) supplied as the base are printed to the string.

If the supplied buffer is NULL, the required length of the string is returned via the string_size parameter.

If the supplied buffer is not NULL, the string representation of the **DDS_DomainParticipantFactoryQos** (p. 967) is written to the buffer.

Parameters

string	*<<**out**>> (p. 433)* The char buffer that will be used to store the string representation of this **DDS_DomainParticipantFactoryQos** (p. 967). If NULL, this function will return the required length of this buffer through the string_size parameter.
string_size	*<<**inout**>> (p. 433)* Cannot be NULL. The size of the supplied buffer. If the supplied buffer is NULL, or if the supplied buffer is not large enough, this value will be updated with the required length of the buffer.
base	*<<**in**>> (p. 433)* The **DDS_DomainParticipantFactoryQos** (p. 967) to be used as the base QoS profile. Only the differences with respect to this base profile will be included in the output string.
format	*<<**in**>> (p. 433)* The **DDS_QosPrintFormat** (p. 1285) to be used to format the output string.

Returns

DDS_RETCODE_OK (p. 532) if no error was encountered.

7.4.4.7 to_string() [5/6]

```
DDS_ReturnCode_t DDS_DomainParticipantFactoryQos::to_string (
            char * string,
             DDS_UnsignedLong & string_size,
            const  DDS_QosPrintAll_t &  ) const  [inline]
```

Obtains a string representation of this **DDS_DomainParticipantFactoryQos** (p. 967).

This overload behaves the same as **DDS_DomainParticipantFactoryQos::to_string** (p. 45) but prints the entire **DDS↩ _DomainParticipantFactoryQos** (p. 967) object. The only valid value for the argument of type **DDS_QosPrintAll_t** (p. 1285) is **DDS_QOS_PRINT_ALL** (p. 169).

For example

```
DomainParticipantFactoryQos qos;
char *str = NULL;
DDS_UnsignedLong str_size = 0;
qos.to_string(str, str_size, QOS_PRINT_ALL);
```

The resultant string is formatted according to the default value for **DDS_QosPrintFormat** (p. 1285).

Parameters

string	<<***out***>> *(p. 433)* The char buffer that will be used to store the string representation of this **DDS_DomainParticipantFactoryQos** (p. 967). If NULL, this function will return the required length of this buffer through the string_size parameter.
string_size	<<***inout***>> *(p. 433)* Cannot be NULL. The size of the supplied buffer. If the supplied buffer is NULL, or if the supplied buffer is not large enough, this value will be updated with the required length of the buffer.

Returns

DDS_RETCODE_OK (p. 532) if no error was encountered.

See also

DDS_DomainParticipantFactoryQos::to_string(char*, DDS_UnsignedLong&, const DDS_DomainParticipantFactoryQos&, const DDS_QosPrintFormat&) const (p. 47)

References **DDS_QosPrintFormat_INITIALIZER**.

7.4.4.8 to_string() [6/6]

```
DDS_ReturnCode_t DDS_DomainParticipantFactoryQos::to_string (
            char * string,
             DDS_UnsignedLong & string_size,
            const  DDS_QosPrintAll_t & ,
            const  DDS_QosPrintFormat & format ) const  [inline]
```

Obtains a string representation of this **DDS_DomainParticipantFactoryQos** (p. 967).

This overload behaves the same as the **DDS_DomainParticipantFactoryQos::to_string(char*, DDS_UnsignedLong&, const DDS_QosPrintAll_t&) const** (p. 48) overload. The only difference is that it allows the caller to specify the **DDS_QosPrintFormat** (p. 1285) which is used to format the resultant string.

Parameters

string	<<***out***>> *(p. 433)* The char buffer that will be used to store the string representation of this **DDS_DomainParticipantFactoryQos** (p. 967). If NULL, this function will return the required length of this buffer through the string_size parameter.
string_size	<<***inout***>> *(p. 433)* Cannot be NULL. The size of the supplied buffer. If the supplied buffer is NULL, or if the supplied buffer is not large enough, this value will be updated with the required length of the buffer.
format	<<***in***>> *(p. 433)* The **DDS_QosPrintFormat** (p. 1285) to be used to format the output string.

Returns

DDS_RETCODE_OK (p. 532) if no error was encountered.

See also

DDS_DomainParticipantFactoryQos::to_string(char*, DDS_UnsignedLong&, const DDS_Domain↩ ParticipantFactoryQos&, const DDS_QosPrintFormat&) const (p. 47)

7.4.5 Variable Documentation

7.4.5.1 DDS_PARTICIPANT_QOS_DEFAULT

```
const struct DDS_DomainParticipantQos DDS_PARTICIPANT_QOS_DEFAULT [extern]
```

Special value for creating a DomainParticipant with default QoS.

When used in **DDSDomainParticipantFactory::create_participant** (p. 1710), this special value is used to indicate that the **DDSDomainParticipant** (p. 1619) should be created with the default **DDSDomainParticipant** (p. 1619) QoS by means of the operation **DDSDomainParticipantFactory::get_default_participant_qos()** (p. 1700) and using the resulting QoS to create the **DDSDomainParticipant** (p. 1619).

When used in **DDSDomainParticipantFactory::set_default_participant_qos** (p. 1698), this special value is used to indicate that the default QoS should be reset back to the initial value that would be used if the **DDSDomainParticipant↩ Factory::set_default_participant_qos** (p. 1698) operation had never been called.

When used in **DDSDomainParticipant::set_qos** (p. 1677), this special value is used to indicate that the QoS of the **DDSDomainParticipant** (p. 1619) should be changed to match the current default QoS set in the **DDSDomain↩ ParticipantFactory** (p. 1694) that the **DDSDomainParticipant** (p. 1619) belongs to.

RTI Connext treats this special value as a constant.

Warning

This value is a constant and should never be modified. You cannot use this value to *get* the default QoS values from the DomainParticipant factory; for this purpose, use **DDSDomainParticipantFactory::get_default_participant↩ _qos** (p. 1700).

See also

NDDS_DISCOVERY_PEERS (p. 662)

DDSDomainParticipantFactory::create_participant() (p. 1710)

DDSDomainParticipantFactory::set_default_participant_qos() (p. 1698)

DDSDomainParticipant::set_qos() (p. 1677)

7.4.5.2 DDS_PARTICIPANT_CONFIG_PARAMS_DEFAULT

```
const struct DDS_DomainParticipantConfigParams_t DDS_PARTICIPANT_CONFIG_PARAMS_DEFAULT [extern]
```

Special value for creating a **DDSDomainParticipant** (p. 1619) from configuration using default parameters.

This value can be used only in **DDSDomainParticipantFactory::create_participant_from_config_w_params** (p. 1718) and indicates that the **DDSDomainParticipant** (p. 1619) must be created applying the information defined in the participant configuration. That is, the domain ID, participant entity name, and QoS profiles for all the entities will be retrieved from the configuration.

RTI Connext treats this special value as a constant.

See also

DDS_DomainParticipantConfigParams_t (p. 964)

DDSDomainParticipantFactory::create_participant_from_config_w_params (p. 1718)

7.5 DomainParticipants

DDSDomainParticipant (p. 1619) entity and associated elements

Classes

- struct **DDS_InvalidLocalAdvanceNoticeStatus**

  DDS_INVALID_LOCAL_ADVANCE_NOTICE_STATUS *(p. 543)*
- struct **DDS_DomainParticipantQos**

  *QoS policies supported by a **DDSDomainParticipant** (p. 1619) entity.*
- struct **DDS_DomainParticipantProtocolStatus**

  *<<**extension**>> (p. 432) The status of a participant's protocol related metrics, like the number of corrupted messages, change in corrupted messages and timestamp of the last corrupted message.*
- class **DDSDomainParticipantListener**

  *<<**interface**>> (p. 433) Listener for participant status.*
- class **DDSDomainParticipant**

  *<<**interface**>> (p. 433) Container for all **DDSDomainEntity** (p. 1618) objects.*

Typedefs

- typedef DDS_DOMAINID_TYPE_NATIVE **DDS_DomainId_t**

  *An integer that indicates in which domain a **DDSDomainParticipant** (p. 1619) communicates.*

Functions

- **DDS_Boolean DDS_DomainParticipantQos_equals** (const struct **DDS_DomainParticipantQos** *self, const struct **DDS_DomainParticipantQos** *other)

  *Compares two **DDS_DomainParticipantQos** (p. 971) for equality.*
- **DDS_ReturnCode_t DDS_DomainParticipantQos::print** () const

  *Prints this **DDS_DomainParticipantQos** (p. 971) to stdout.*
- **DDS_ReturnCode_t DDS_DomainParticipantQos::to_string** (char *string, **DDS_UnsignedLong** &string_size) const

  *Obtains a string representation of this **DDS_DomainParticipantQos** (p. 971).*
- **DDS_ReturnCode_t DDS_DomainParticipantQos::to_string** (char *string, **DDS_UnsignedLong** &string_size, const **DDS_DomainParticipantQos** &base) const

  *Obtains a string representation of this **DDS_DomainParticipantQos** (p. 971).*
- **DDS_ReturnCode_t DDS_DomainParticipantQos::to_string** (char *string, **DDS_UnsignedLong** &string_size, const **DDS_QosPrintFormat** &format) const

  *Obtains a string representation of this **DDS_DomainParticipantQos** (p. 971).*
- **DDS_ReturnCode_t DDS_DomainParticipantQos::to_string** (char *string, **DDS_UnsignedLong** &string_size, const **DDS_DomainParticipantQos** &base, const **DDS_QosPrintFormat** &format) const

  *Obtains a string representation of this **DDS_DomainParticipantQos** (p. 971).*
- **DDS_ReturnCode_t DDS_DomainParticipantQos::to_string** (char *string, **DDS_UnsignedLong** &string_size, const **DDS_QosPrintAll_t** &) const

  *Obtains a string representation of this **DDS_DomainParticipantQos** (p. 971).*
- **DDS_ReturnCode_t DDS_DomainParticipantQos::to_string** (char *string, **DDS_UnsignedLong** &string_size, const **DDS_QosPrintAll_t** &, const **DDS_QosPrintFormat** &format) const

  *Obtains a string representation of this **DDS_DomainParticipantQos** (p. 971).*

Variables

- const struct **DDS_TopicQos DDS_TOPIC_QOS_DEFAULT**

  *Special value for creating a **DDSTopic** (p. 1885) with default QoS.*
- const struct **DDS_PublisherQos DDS_PUBLISHER_QOS_DEFAULT**

  *Special value for creating a **DDSPublisher** (p. 1818) with default QoS.*
- const struct **DDS_SubscriberQos DDS_SUBSCRIBER_QOS_DEFAULT**

  *Special value for creating a **DDSSubscriber** (p. 1860) with default QoS.*
- const struct **DDS_FlowControllerProperty_t DDS_FLOW_CONTROLLER_PROPERTY_DEFAULT**

  *<<**extension**>> (p. 432) Special value for creating a **DDSFlowController** (p. 1736) with default property.*
- const char *const **DDS_SQLFILTER_NAME**

  *<<**extension**>> (p. 432) The name of the built-in SQL filter that can be used with ContentFilteredTopics and MultiChannel DataWriters.*
- const char *const **DDS_STRINGMATCHFILTER_NAME**

  *<<**extension**>> (p. 432) The name of the built-in StringMatch filter that can be used with ContentFilteredTopics and MultiChannel DataWriters.*

7.5.1 Detailed Description

DDSDomainParticipant (p. 1619) entity and associated elements

7.5.2 Typedef Documentation

7.5.2.1 DDS_DomainId_t

```
typedef DDS_DOMAINID_TYPE_NATIVE  DDS_DomainId_t
```

An integer that indicates in which domain a **DDSDomainParticipant** (p. 1619) communicates.

Participants with the same **DDS_DomainId_t** (p. 53) are said to be in the same domain, and can thus communicate with one another.

The lower limit for a domain ID is 0. The upper limit for a domain ID is determined by the guidelines stated in **DDS_RtpsWellKnownPorts_t** (p. 1338) (specifically **DDS_RtpsWellKnownPorts_t::domain_id_gain** (p. 1341))

7.5.3 Function Documentation

7.5.3.1 DDS_DomainParticipantQos_equals()

```
DDS_Boolean DDS_DomainParticipantQos_equals (
            const struct  DDS_DomainParticipantQos * self,
            const struct  DDS_DomainParticipantQos * other )
```

Compares two **DDS_DomainParticipantQos** (p. 971) for equality.

Parameters

self	<<***in***>> *(p. 433)* This DomainParticipantQos.
other	<<***in***>> *(p. 433)* The other DomainParticipantQos to be compared with this DomainParticipantQos.

Returns

DDS_BOOLEAN_TRUE (p. 513) if the two Qos objects are equal or **DDS_BOOLEAN_FALSE** (p. 513) otherwise.

Referenced by **DDS_DomainParticipantQos::operator!=()**, and **DDS_DomainParticipantQos::operator==()**.

7.5.3.2 print()

```
DDS_ReturnCode_t DDS_DomainParticipantQos::print ( ) const  [inline]
```

Prints this **DDS_DomainParticipantQos** (p. 971) to stdout.

Only the differences between this **DDS_DomainParticipantQos** (p. 971) and the documented default are printed. If you wish to print everything regardless, see **DDS_DomainParticipantQos::to_string(char*, DDS_UnsignedLong&, const DDS_DomainParticipantQos&, const DDS_QosPrintFormat&) const** (p. 56). The output is formatted according to the default values for **DDS_QosPrintFormat** (p. 1285).

Returns

DDS_RETCODE_OK (p. 532) if no error was encountered.

7.5.3.3 to_string() [1/6]

```
DDS_ReturnCode_t DDS_DomainParticipantQos::to_string (
            char * string,
             DDS_UnsignedLong & string_size ) const  [inline]
```

Obtains a string representation of this **DDS_DomainParticipantQos** (p. 971).

Only the differences between this **DDS_DomainParticipantQos** (p. 971) and the documented default are printed. If you wish to print everything regardless, see **DDS_DomainParticipantQos::to_string(char*, DDS_UnsignedLong&, const DDS_DomainParticipantQos&, const DDS_QosPrintFormat&) const** (p. 56). The output is formatted according to the default values for **DDS_QosPrintFormat** (p. 1285).

If the supplied buffer is NULL, the required length of the string is returned via the string_size parameter.

If the supplied buffer is not NULL, the string representation of the **DDS_DomainParticipantQos** (p. 971) is written to the buffer.

Parameters

string	<<***out***>> *(p. 433)* The char buffer that will be used to store the string representation of this **DDS_DomainParticipantQos** (p. 971). If NULL, this function will return the required length of this buffer through the string_size parameter.
string_size	<<***inout***>> *(p. 433)* Cannot be NULL. The size of the supplied buffer. If the supplied buffer is NULL, or if the supplied buffer is not large enough, this value will be updated with the required length of the buffer.

Returns

DDS_RETCODE_OK (p. 532) if no error was encountered.

See also

DDS_DomainParticipantQos::to_string(char*, DDS_UnsignedLong&, const DDS_DomainParticipantQos&, const DDS_QosPrintFormat&) const (p. 56)

7.5.3.4 to_string() [2/6]

```
DDS_ReturnCode_t DDS_DomainParticipantQos::to_string (
            char * string,
             DDS_UnsignedLong & string_size,
            const  DDS_DomainParticipantQos & base ) const  [inline]
```

Obtains a string representation of this **DDS_DomainParticipantQos** (p. 971).

This overload behaves the same as **DDS_DomainParticipantQos::to_string** (p. 54) but allows the caller to specify the **DDS_DomainParticipantQos** (p. 971), which is used as the base profile. Only the differences between the QoS and the base profile are included in the output string.

Parameters

string	<<***out***>> *(p. 433)* The char buffer that will be used to store the string representation of this **DDS_DomainParticipantQos** (p. 971). If NULL, this function will return the required length of this buffer through the string_size parameter.
string_size	<<***inout***>> *(p. 433)* Cannot be NULL. The size of the supplied buffer. If the supplied buffer is NULL, or if the supplied buffer is not large enough, this value will be updated with the required length of the buffer.
base	<<***in***>> *(p. 433)* The **DDS_DomainParticipantQos** (p. 971) to be used as the base QoS profile. Only the differences with respect to this base profile will be included in the output string.

Returns

DDS_RETCODE_OK (p. 532) if no error was encountered.

See also

DDS_DomainParticipantQos::to_string(char*, DDS_UnsignedLong&, const DDS_DomainParticipant↩ Qos&, const DDS_QosPrintFormat&) const (p. 56)

References **DDS_QosPrintFormat_INITIALIZER**.

7.5.3.5 to_string() [3/6]

```
DDS_ReturnCode_t DDS_DomainParticipantQos::to_string (
            char * string,
             DDS_UnsignedLong & string_size,
            const  DDS_QosPrintFormat & format ) const  [inline]
```

Obtains a string representation of this **DDS_DomainParticipantQos** (p. 971).

This overload behaves the same as **DDS_DomainParticipantQos::to_string** (p. 54) but allows the caller to specify the **DDS_QosPrintFormat** (p. 1285) which is used to format the output.

Parameters

string	*<<**out**>> (p. 433)* The char buffer that will be used to store the string representation of this **DDS_DomainParticipantQos** (p. 971). If NULL, this function will return the required length of this buffer through the string_size parameter.
string_size	*<<**inout**>> (p. 433)* Cannot be NULL. The size of the supplied buffer. If the supplied buffer is NULL, or if the supplied buffer is not large enough, this value will be updated with the required length of the buffer.
format	*<<**in**>> (p. 433)* The **DDS_QosPrintFormat** (p. 1285) to be used to format the output string.

Returns

DDS_RETCODE_OK (p. 532) if no error was encountered.

See also

DDS_DomainParticipantQos::to_string(char*, DDS_UnsignedLong&, const DDS_DomainParticipantQos&, const DDS_QosPrintFormat&) const (p. 56)

References **DDS_RETCODE_ERROR**, and **DDS_RETCODE_OK**.

7.5.3.6 to_string() [4/6]

```
DDS_ReturnCode_t DDS_DomainParticipantQos::to_string (
            char * string,
             DDS_UnsignedLong & string_size,
            const  DDS_DomainParticipantQos & base,
            const  DDS_QosPrintFormat & format ) const  [inline]
```

Obtains a string representation of this **DDS_DomainParticipantQos** (p. 971).

Only the differences between this **DDS_DomainParticipantQos** (p. 971) and the **DDS_DomainParticipantQos** (p. 971) supplied as the base are printed to the string.

If the supplied buffer is NULL, the required length of the string is returned via the string_size parameter.

If the supplied buffer is not NULL, the string representation of the **DDS_DomainParticipantQos** (p. 971) is written to the buffer.

Parameters

string	*<<**out**>> (p. 433)* The char buffer that will be used to store the string representation of this **DDS_DomainParticipantQos** (p. 971). If NULL, this function will return the required length of this buffer through the string_size parameter.
string_size	*<<**inout**>> (p. 433)* Cannot be NULL. The size of the supplied buffer. If the supplied buffer is NULL, or if the supplied buffer is not large enough, this value will be updated with the required length of the buffer.
base	*<<**in**>> (p. 433)* The **DDS_DomainParticipantQos** (p. 971) to be used as the base QoS profile.
	Only the differences with respect to this base profile will be included in the output string.
format	*<<**in**>> (p. 433)* The **DDS_QosPrintFormat** (p. 1285) to be used to format the output string.

Returns

DDS_RETCODE_OK (p. 532) if no error was encountered.

7.5.3.7 to_string() [5/6]

```
DDS_ReturnCode_t DDS_DomainParticipantQos::to_string (
            char * string,
             DDS_UnsignedLong & string_size,
            const  DDS_QosPrintAll_t &  ) const  [inline]
```

Obtains a string representation of this **DDS_DomainParticipantQos** (p. 971).

This overload behaves the same as **DDS_DomainParticipantQos::to_string** (p. 54) but prints the entire **DDS_↩ DomainParticipantQos** (p. 971) object. The only valid value for the argument of type **DDS_QosPrintAll_t** (p. 1285) is **DDS_QOS_PRINT_ALL** (p. 169).

For example
```
DomainParticipantQos qos;
char *str = NULL;
DDS_UnsignedLong str_size = 0;
qos.to_string(str, str_size, QOS_PRINT_ALL);
```

The resultant string is formatted according to the default value for **DDS_QosPrintFormat** (p. 1285).

Parameters

string	<<***out***>> *(p. 433)* The char buffer that will be used to store the string representation of this **DDS_DomainParticipantQos** (p. 971). If NULL, this function will return the required length of this buffer through the string_size parameter.
string_size	<<***inout***>> *(p. 433)* Cannot be NULL. The size of the supplied buffer. If the supplied buffer is NULL, or if the supplied buffer is not large enough, this value will be updated with the required length of the buffer.

Returns

DDS_RETCODE_OK (p. 532) if no error was encountered.

See also

DDS_DomainParticipantQos::to_string(char∗, DDS_UnsignedLong&, const DDS_DomainParticipant↩ Qos&, const DDS_QosPrintFormat&) const (p. 56)

References **DDS_QosPrintFormat_INITIALIZER**.

7.5.3.8 to_string() [6/6]

```
DDS_ReturnCode_t DDS_DomainParticipantQos::to_string (
            char * string,
             DDS_UnsignedLong & string_size,
            const  DDS_QosPrintAll_t & ,
            const  DDS_QosPrintFormat & format ) const  [inline]
```

Obtains a string representation of this **DDS_DomainParticipantQos** (p. 971).

This overload behaves the same as the **DDS_DomainParticipantQos::to_string(char*, DDS_UnsignedLong&, const DDS_QosPrintAll_t&) const** (p. 57) overload. The only difference is that it allows the caller to specify the **DDS_QosPrintFormat** (p. 1285) which is used to format the resultant string.

Parameters

string	<<***out***>> *(p. 433)* The char buffer that will be used to store the string representation of this **DDS_DomainParticipantQos** (p. 971). If NULL, this function will return the required length of this buffer through the string_size parameter.
string_size	<<***inout***>> *(p. 433)* Cannot be NULL. The size of the supplied buffer. If the supplied buffer is NULL, or if the supplied buffer is not large enough, this value will be updated with the required length of the buffer.
format	<<***in***>> *(p. 433)* The **DDS_QosPrintFormat** (p. 1285) to be used to format the output string.

Returns

DDS_RETCODE_OK (p. 532) if no error was encountered.

See also

DDS_DomainParticipantQos::to_string(char*, DDS_UnsignedLong&, const DDS_DomainParticipant↩ Qos&, const DDS_QosPrintFormat&) const (p. 56)

7.5.4 Variable Documentation

7.5.4.1 DDS_TOPIC_QOS_DEFAULT

```
const struct  DDS_TopicQos DDS_TOPIC_QOS_DEFAULT  [extern]
```

Special value for creating a **DDSTopic** (p. 1885) with default QoS.

When used in **DDSDomainParticipant::create_topic** (p. 1650), this special value is used to indicate that the **DDSTopic** (p. 1885) should be created with the default **DDSTopic** (p. 1885) QoS by means of the operation get_default_topic_qos and using the resulting QoS to create the **DDSTopic** (p. 1885).

When used in **DDSDomainParticipant::set_default_topic_qos** (p. 1637), this special value is used to indicate that the default QoS should be reset back to the initial value that would be used if the **DDSDomainParticipant::set_default_↩ topic_qos** (p. 1637) operation had never been called.

When used in **DDSTopic::set_qos** (p. 1887), this special value is used to indicate that the QoS of the **DDSTopic** (p. 1885) should be changed to match the current default QoS set in the **DDSDomainParticipant** (p. 1619) that the **DDSTopic** (p. 1885) belongs to.

Warning

This value is a constant and should never be modified. You cannot use this value to *get* the default QoS values for a Topic; for this purpose, use **DDSDomainParticipant::get_default_topic_qos** (p. 1636).

See also

DDSDomainParticipant::create_topic (p. 1650)

DDSDomainParticipant::set_default_topic_qos (p. 1637)

DDSTopic::set_qos (p. 1887)

Examples

HelloWorld_publisher.cxx, and **HelloWorld_subscriber.cxx**.

7.5.4.2 DDS_PUBLISHER_QOS_DEFAULT

```
const struct DDS_PublisherQos DDS_PUBLISHER_QOS_DEFAULT [extern]
```

Special value for creating a **DDSPublisher** (p. 1818) with default QoS.

When used in **DDSDomainParticipant::create_publisher** (p. 1644), this special value is used to indicate that the **DDSPublisher** (p. 1818) should be created with the default **DDSPublisher** (p. 1818) QoS by means of the operation get_default_publisher_qos and using the resulting QoS to create the **DDSPublisher** (p. 1818).

When used in **DDSDomainParticipant::set_default_publisher_qos** (p. 1640), this special value is used to indicate that the default QoS should be reset back to the initial value that would be used if the **DDSDomainParticipant::set_↩ default_publisher_qos** (p. 1640) operation had never been called.

When used in **DDSPublisher::set_qos** (p. 1836), this special value is used to indicate that the QoS of the **DDSPublisher** (p. 1818) should be changed to match the current default QoS set in the **DDSDomainParticipant** (p. 1619) that the **DDSPublisher** (p. 1818) belongs to.

Warning

This value is a constant and should never be modified. You cannot use this value to *get* the default QoS values for a Publisher; for this purpose, use **DDSDomainParticipant::get_default_publisher_qos** (p. 1639).

See also

DDSDomainParticipant::create_publisher (p. 1644)

DDSDomainParticipant::set_default_publisher_qos (p. 1640)

DDSPublisher::set_qos (p. 1836)

Examples

HelloWorld_publisher.cxx.

7.5.4.3 DDS_SUBSCRIBER_QOS_DEFAULT

```
const struct DDS_SubscriberQos DDS_SUBSCRIBER_QOS_DEFAULT [extern]
```

Special value for creating a **DDSSubscriber** (p. 1860) with default QoS.

When used in **DDSDomainParticipant::create_subscriber** (p. 1647), this special value is used to indicate that the **DDSSubscriber** (p. 1860) should be created with the default **DDSSubscriber** (p. 1860) QoS by means of the operation get_default_subscriber_qos and using the resulting QoS to create the **DDSSubscriber** (p. 1860).

When used in **DDSDomainParticipant::set_default_subscriber_qos** (p. 1642), this special value is used to indicate that the default QoS should be reset back to the initial value that would be used if the **DDSDomainParticipant::set_default_subscriber_qos** (p. 1642) operation had never been called.

When used in **DDSSubscriber::set_qos** (p. 1877), this special value is used to indicate that the QoS of the **DDSSubscriber** (p. 1860) should be changed to match the current default QoS set in the **DDSDomainParticipant** (p. 1619) that the **DDSSubscriber** (p. 1860) belongs to.

Warning

This value is a constant and should never be modified. You cannot use this value to *get* the default QoS values for a Subscriber; for this purpose, use **DDSDomainParticipant::get_default_subscriber_qos** (p. 1641).

See also

DDSDomainParticipant::create_subscriber (p. 1647)

DDSDomainParticipant::get_default_subscriber_qos (p. 1641)

DDSSubscriber::set_qos (p. 1877)

Examples

HelloWorld_subscriber.cxx.

7.5.4.4 DDS_FLOW_CONTROLLER_PROPERTY_DEFAULT

```
const struct DDS_FlowControllerProperty_t DDS_FLOW_CONTROLLER_PROPERTY_DEFAULT [extern]
```

<<***extension***>> *(p. 432)* Special value for creating a **DDSFlowController** (p. 1736) with default property.

When used in **DDSDomainParticipant::create_flowcontroller** (p. 1659) and **DDSFlowController::set_property** (p. 1737), this special value represents the set of values specified on the last successful call to **DDSDomainParticipant::set_default_flowcontroller_property** (p. 1631), or else, if the call was never made, the default values listed in **DDS_FlowControllerProperty_t** (p. 1156).

When used in **DDSDomainParticipant::set_default_flowcontroller_property** (p. 1631), this special value indicates that the default QoS should be reset back to the initial value that would be used if the **DDSDomainParticipant::set_default_flowcontroller_property** (p. 1631) operation had never been called.

Warning

This value is a constant and should never be modified. You cannot use this value to *get* the default properties for a FlowController; for this purpose, use **DDSDomainParticipant::get_default_flowcontroller_property** (p. 1630).

See also

DDSDomainParticipant::create_flowcontroller (p. 1659)

DDSDomainParticipant::set_default_flowcontroller_property (p. 1631)

DDSFlowController::set_property (p. 1737)

7.5.4.5 DDS_SQLFILTER_NAME

```
const char* const DDS_SQLFILTER_NAME
```

<<***extension***>> *(p. 432)* The name of the built-in SQL filter that can be used with ContentFilteredTopics and Multi↩ Channel DataWriters.

See also

Queries and Filters Syntax (p. 184)

7.5.4.6 DDS_STRINGMATCHFILTER_NAME

```
const char* const DDS_STRINGMATCHFILTER_NAME
```

<<***extension***>> *(p. 432)* The name of the built-in StringMatch filter that can be used with ContentFilteredTopics and MultiChannel DataWriters.

The StringMatch Filter is a subset of the SQL filter; it only supports the MATCH relational operator on a single string field.

See also

Queries and Filters Syntax (p. 184)

7.6 Built-in Topics

Built-in objects created by RTI Connext but accessible to the application.

Modules

- **Participant Built-in Topics**

  Builtin topic for accessing information about the DomainParticipants discovered by RTI Connext.
- **Topic Built-in Topics**
- **Publication Built-in Topics**

  Builtin topic for accessing information about the Publications discovered by RTI Connext.
- **Subscription Built-in Topics**

  Builtin topic for accessing information about the Subscriptions discovered by RTI Connext.
- **ServiceRequest Built-in Topic**

  Builtin topic for accessing requests from different services within RTI Connext.
- **Common types and functions**

  Types and functions related to the built-in topics.

7.6.1 Detailed Description

Built-in objects created by RTI Connext but accessible to the application.

RTI Connext must discover and keep track of the remote entities, such as new participants in the domain. This information may also be important to the application, which may want to react to this discovery, or else access it on demand.

A set of built-in topics and corresponding **DDSDataReader** (p. 1556) objects are introduced to be used by the application to access this discovery information.

The information can be accessed as if it were normal application data. This allows the application to know when there are any changes in those values by means of the **DDSListener** (p. 1794) or the **DDSCondition** (p. 1544) mechanisms.

The built-in data-readers all belong to a built-in **DDSSubscriber** (p. 1860), which can be retrieved by using the method **DDSDomainParticipant::get_builtin_subscriber** (p. 1661). The built-in **DDSDataReader** (p. 1556) objects can be retrieved by using the operation **DDSSubscriber::lookup_datareader** (p. 1872), with the topic name as a parameter.

Built-in entities have default listener settings as well. The built-in **DDSSubscriber** (p. 1860) and all of its built-in topics have 'nil' listeners with all statuses appearing in their listener masks (acting as a NO-OP listener that does not reset communication status). The built-in DataReaders have null listeners with no statuses in their masks.

The information that is accessible about the remote entities by means of the built-in topics includes all the QoS policies that apply to the corresponding remote Entity. These QoS policies appear as normal 'data' fields inside the data read by means of the built-in Topic. Additional information is provided to identify the Entity and facilitate the application logic.

The built-in **DDSDataReader** (p. 1556) will not provide data pertaining to entities created from the same **DDSDomain↩ Participant** (p. 1619) under the assumption that such entities are already known to the application that created them.

The discovery information provided by each built-in reader won't be received until it is first looked up. And if the Domain↩ Participant was created in a disabled state (see **DDS_EntityFactoryQosPolicy** (p. 1145)), the built-in subscriber and readers will need to be enabled explicitly (by the **DDSEntity::enable** (p. 1734) operation).

Refer to **DDS_ParticipantBuiltinTopicData** (p. 1229), **DDS_TopicBuiltinTopicData** (p. 1391), **DDS_Subscription↩ BuiltinTopicData** (p. 1370) and **DDS_PublicationBuiltinTopicData** (p. 1264) for a description of all the built-in topics and their contents.

The QoS of the built-in **DDSSubscriber** (p. 1860) and **DDSDataReader** (p. 1556) objects is given by the following table:

Table 7.18 ***QoS of built-in DDSSubscriber (*** p. ***1860) and DDSDataReader (*** p. ***1556)***

QoS	Value
DDS_UserDataQosPolicy (p. 1504)	0-length sequence
DDS_TopicDataQosPolicy (p. 1396)	0-length sequence
DDS_GroupDataQosPolicy (p. 1160)	0-length sequence
DDS_DurabilityQosPolicy (p. 1005)	**DDS_TRANSIENT_LOCAL_DURABILITY_QOS** (p. 599)
DDS_DurabilityServiceQosPolicy (p. 1009)	Does not apply as **DDS_DurabilityQosPolicyKind** (p. 598) is **DDS_TRANSIENT_LOCAL_DURABILITY_QOS** (p. 599)
DDS_PresentationQosPolicy (p. 1249)	access_scope = **DDS_TOPIC_PRESENTATION_QOS** (p. 616) coherent_access = **DDS_BOOLEAN_FALSE** (p. 513) ordered_access = **DDS_BOOLEAN_FALSE** (p. 513)
DDS_DeadlineQosPolicy (p. 928)	Period = infinite
DDS_LatencyBudgetQosPolicy (p. 1173)	duration = 0
DDS_OwnershipQosPolicy (p. 1222)	**DDS_SHARED_OWNERSHIP_QOS** (p. 613)
DDS_OwnershipStrengthQosPolicy (p. 1227)	value = 0
DDS_LivelinessQosPolicy (p. 1180)	kind = **DDS_AUTOMATIC_LIVELINESS_QOS** (p. 608) lease_duration = 0
DDS_TimeBasedFilterQosPolicy (p. 1389)	minimum_separation = 0
DDS_PartitionQosPolicy (p. 1240)	0-length sequence
DDS_ReliabilityQosPolicy (p. 1295)	kind = **DDS_RELIABLE_RELIABILITY_QOS** (p. 633) max_blocking_time = 100 milliseconds
DDS_DestinationOrderQosPolicy (p. 931)	**DDS_BY_RECEPTION_TIMESTAMP_↩ DESTINATIONORDER_QOS** (p. 585)
DDS_HistoryQosPolicy (p. 1162)	kind = **DDS_KEEP_LAST_HISTORY_QOS** (p. 604) depth = 1
DDS_ResourceLimitsQosPolicy (p. 1308)	max_samples = **DDS_LENGTH_UNLIMITED** (p. 636) max_instances = **DDS_LENGTH_UNLIMITED** (p. 636) max_samples_per_instance = **DDS_LENGTH_UNLIMITED** (p. 636)
DDS_ReaderDataLifecycleQosPolicy (p. 1289)	autopurge_nowriter_samples_delay = infinite autopurge_disposed_samples_delay = infinite
DDS_EntityFactoryQosPolicy (p. 1145)	autoenable_created_entities = **DDS_BOOLEAN_TRUE** (p. 513)

7.7 Topic Module

Contains the **DDSTopic** (p. 1885), **DDSContentFilteredTopic** (p. 1551), and **DDSMultiTopic** (p. 1797) classes, the **DDSTopicListener** (p. 1894) interface, and more generally, all that is needed by an application to define **DDSTopic** (p. 1885) objects and attach QoS policies to them.

Modules

- **Topics**

DDSTopic *(p. 1885) entity and associated elements*

- **Zero Copy Transfer Over Shared Memory**

  *<<**extension**>> (p. 432) Zero Copy transfer over shared memory*
- **User Data Type Support**

  Defines generic classes and macros to support user data types.
- **Type Code Support**

  *<<**extension**>> (p. 432) A **DDS_TypeCode** (p. 1428) is a mechanism for representing a type at runtime. RTI Connext can use type codes to send type definitions on the network. You will need to understand this API in order to use the **Dynamic Data** (p. 103) capability or to inspect the type information you receive from remote readers and writers.*
- **Built-in Types**

  RTI Connext provides a set of very simple data types for you to use with the topics in your application.
- **Built-in Topic's Trust Types**

  *Types used as part of **DDS_ParticipantBuiltinTopicData** (p. 1229), **DDS_PublicationBuiltinTopicData** (p. 1264), **DDS_SubscriptionBuiltinTopicData** (p. 1370) to describe Trust Plugins configuration.*
- **Dynamic Data**

  *<<**extension**>> (p. 432) The Dynamic Data API provides a way to interact with arbitrarily complex data types at runtime without the need for code generation.*
- **DDS-Specific Primitive Types**

  Basic DDS value types for use in user data types.
- **FlatData Topic-Types**

  *<<**extension**>> (p. 432) FlatData Language Binding for IDL topic-types*

7.7.1 Detailed Description

Contains the **DDSTopic** (p. 1885), **DDSContentFilteredTopic** (p. 1551), and **DDSMultiTopic** (p. 1797) classes, the **DDSTopicListener** (p. 1894) interface, and more generally, all that is needed by an application to define **DDSTopic** (p. 1885) objects and attach QoS policies to them.

7.8 Topics

DDSTopic (p. 1885) entity and associated elements

Classes

- struct **DDS_PrintFormatProperty**

  A collection of attributes used to configure how data samples will be formatted when converted to a string.
- struct **DDS_InconsistentTopicStatus**

  ***DDS_INCONSISTENT_TOPIC_STATUS** (p. 539)*
- struct **DDS_TopicQos**

  *QoS policies supported by a **DDSTopic** (p. 1885) entity.*
- struct **DDS_ExpressionProperty**

  *Provides additional information about the filter expression passed to **DDSWriterContentFilter::writer_compile** (p. 1906) .*
- struct **DDS_FilterSampleInfo**

  Provides meta information associated with the sample.

- class **DDSTopicDescription**

  *<<**interface**>> (p. 433) Base class for **DDSTopic** (p. 1885), **DDSContentFilteredTopic** (p. 1551), and **DDSMultiTopic** (p. 1797).*
- class **DDSContentFilteredTopic**

  *<<**interface**>> (p. 433) Specialization of **DDSTopicDescription** (p. 1892) that allows for content-based subscriptions.*
- class **DDSMultiTopic**

  ***[Not supported (optional)]** <<**interface**>> (p. 433) A specialization of **DDSTopicDescription** (p. 1892) that allows subscriptions that combine/filter/rearrange data coming from several topics.*
- class **DDSTopic**

  *<<**interface**>> (p. 433) The most basic description of the data to be published and subscribed.*
- class **DDSTopicListener**

  *<<**interface**>> (p. 433) **DDSListener** (p. 1794) for **DDSTopic** (p. 1885) entities.*
- class **DDSContentFilter**

  *<<**interface**>> (p. 433) Interface to be used by a custom filter of a **DDSContentFilteredTopic** (p. 1551)*
- class **DDSWriterContentFilter**

  *<<**interface**>> (p. 433) Interface to be used by a custom filter of a **DDSContentFilteredTopic** (p. 1551).*

Typedefs

- typedef struct **DDS_PrintFormatProperty** **DDS_PrintFormatProperty**

  A collection of attributes used to configure how data samples will be formatted when converted to a string.

Enumerations

- enum **DDS_PrintFormatKind** {
  DDS_DEFAULT_PRINT_FORMAT ,
  DDS_XML_PRINT_FORMAT ,
  DDS_JSON_PRINT_FORMAT }

  Format kinds available when converting data samples to string representations.

Functions

- **DDS_Boolean** **DDS_TopicQos_equals** (const struct **DDS_TopicQos** *self, const struct **DDS_TopicQos** *other)

  *Compares two **DDS_TopicQos** (p. 1397) for equality.*
- **DDS_ReturnCode_t** **DDS_TopicQos::print** () const

  *Prints this **DDS_TopicQos** (p. 1397) to stdout.*
- **DDS_ReturnCode_t** **DDS_TopicQos::to_string** (char *string, **DDS_UnsignedLong** &string_size) const

  *Obtains a string representation of this **DDS_TopicQos** (p. 1397).*
- **DDS_ReturnCode_t** **DDS_TopicQos::to_string** (char *string, **DDS_UnsignedLong** &string_size, const **DDS_TopicQos** &base) const

  *Obtains a string representation of this **DDS_TopicQos** (p. 1397).*
- **DDS_ReturnCode_t** **DDS_TopicQos::to_string** (char *string, **DDS_UnsignedLong** &string_size, const **DDS_QosPrintFormat** &format) const

  *Obtains a string representation of this **DDS_TopicQos** (p. 1397).*

- **DDS_ReturnCode_t DDS_TopicQos::to_string** (char *string, **DDS_UnsignedLong** &string_size, const **DDS_TopicQos** &base, const **DDS_QosPrintFormat** &format) const

  *Obtains a string representation of this **DDS_TopicQos*** (p. *1397*).
- **DDS_ReturnCode_t DDS_TopicQos::to_string** (char *string, **DDS_UnsignedLong** &string_size, const **DDS_QosPrintAll_t** &) const

  *Obtains a string representation of this **DDS_TopicQos*** (p. *1397*).
- **DDS_ReturnCode_t DDS_TopicQos::to_string** (char *string, **DDS_UnsignedLong** &string_size, const **DDS_QosPrintAll_t** &, const **DDS_QosPrintFormat** &format) const

  *Obtains a string representation of this **DDS_TopicQos*** (p. *1397*).

Variables

- const struct **DDS_PrintFormatProperty DDS_PRINT_FORMAT_PROPERTY_DEFAULT**

  *Sentinel constant indicating default values for **DDS_PrintFormatProperty*** (p. *1253*).
- const struct **DDS_FilterSampleInfo DDS_FILTERSAMPLEINFO_DEFAULT**

  *Initializer for new **DDS_FilterSampleInfo*** (p. *1153*).

7.8.1 Detailed Description

DDSTopic (p. 1885) entity and associated elements

7.8.2 Typedef Documentation

7.8.2.1 DDS_PrintFormatProperty

`typedef struct` **`DDS_PrintFormatProperty DDS_PrintFormatProperty`**

A collection of attributes used to configure how data samples will be formatted when converted to a string.

7.8.3 Enumeration Type Documentation

7.8.3.1 DDS_PrintFormatKind

`enum` **`DDS_PrintFormatKind`**

Format kinds available when converting data samples to string representations.

Enumerator

DDS_DEFAULT_PRINT_FORMAT	Use a default format specific to RTI Connext to represent the data when converting to a string.
DDS_XML_PRINT_FORMAT	Use an XML format to represent the data when converting to a string.
DDS_JSON_PRINT_FORMAT	Use a JSON format to represent the data when converting to a string.

7.8.4 Function Documentation

7.8.4.1 DDS_TopicQos_equals()

```
DDS_Boolean DDS_TopicQos_equals (
            const struct  DDS_TopicQos * self,
            const struct  DDS_TopicQos * other )
```

Compares two **DDS_TopicQos** (p. 1397) for equality.

Parameters

self	<<***in***>> *(p. 433)* This TopicQos.
other	<<***in***>> *(p. 433)* The other TopicQos to be compared with this TopicQos

Returns

DDS_BOOLEAN_TRUE (p. 513) if the two Qos objects are equal or **DDS_BOOLEAN_FALSE** (p. 513) otherwise.

Referenced by **DDS_TopicQos::operator!=()**, and **DDS_TopicQos::operator==()**.

7.8.4.2 print()

```
DDS_ReturnCode_t DDS_TopicQos::print ( ) const  [inline]
```

Prints this **DDS_TopicQos** (p. 1397) to stdout.

Only the differences between this **DDS_TopicQos** (p. 1397) and the documented default are printed. If you wish to print everything regardless, see **DDS_TopicQos::to_string(char*, DDS_UnsignedLong&, const DDS_TopicQos&, const DDS_QosPrintFormat&) const** (p. 70). The output is formatted according to the default values for **DDS_QosPrintFormat** (p. 1285).

Returns

DDS_RETCODE_OK (p. 532) if no error was encountered.

7.8.4.3 to_string() [1/6]

```
DDS_ReturnCode_t DDS_TopicQos::to_string (
            char * string,
             DDS_UnsignedLong & string_size ) const  [inline]
```

Obtains a string representation of this **DDS_TopicQos** (p. 1397).

Only the differences between this **DDS_TopicQos** (p. 1397) and the documented default are printed to the string. If you wish to print everything regardless, see **DDS_TopicQos::to_string(char*, DDS_UnsignedLong&, const DDS_TopicQos&, const DDS_QosPrintFormat&) const** (p. 70). The output is formatted according to the default values for **DDS_QosPrintFormat** (p. 1285).

If the supplied buffer is NULL, the required length of the string is returned via the string_size parameter.

If the supplied buffer is not NULL, the string representation of the **DDS_TopicQos** (p. 1397) is written to the buffer.

Parameters

string	<<***out***>> *(p. 433)* The char buffer that will be used to store the string representation of this **DDS_TopicQos** (p. 1397). If NULL, this function will return the required length of this buffer through the string_size parameter.
string_size	<<***inout***>> *(p. 433)* Cannot be NULL. The size of the supplied buffer. If the supplied buffer is NULL, or if the supplied buffer is not large enough, this value will be updated with the required length of the buffer.

Returns

DDS_RETCODE_OK (p. 532) if no error was encountered.

See also

DDS_TopicQos::to_string(char*, DDS_UnsignedLong&, const DDS_TopicQos&, const DDS_QosPrintFormat&) const (p. 70)

7.8.4.4 to_string() [2/6]

```
DDS_ReturnCode_t DDS_TopicQos::to_string (
            char * string,
             DDS_UnsignedLong & string_size,
            const  DDS_TopicQos & base ) const  [inline]
```

Obtains a string representation of this **DDS_TopicQos** (p. 1397).

This overload behaves the same as **DDS_TopicQos::to_string** (p. 67) but allows the caller to specify the **DDS_TopicQos** (p. 1397), which is used as the base profile. Only the differences between the QoS and the base profile are included in the output string.

Parameters

string	<<***out***>> *(p. 433)* The char buffer that will be used to store the string representation of this **DDS_TopicQos** (p. 1397). If NULL, this function will return the required length of this buffer through the string_size parameter.
string_size	<<***inout***>> *(p. 433)* Cannot be NULL. The size of the supplied buffer. If the supplied buffer is NULL, or if the supplied buffer is not large enough, this value will be updated with the required length of the buffer.
base	<<***in***>> *(p. 433)* The **DDS_TopicQos** (p. 1397) to be used as the base QoS profile. Only the differences with respect to this base profile will be included in the output string.

Returns

DDS_RETCODE_OK (p. 532) if no error was encountered.

See also

DDS_TopicQos::to_string(char*, DDS_UnsignedLong&, const DDS_TopicQos&, const DDS_QosPrint↩ Format&) const (p. 70)

References **DDS_QosPrintFormat_INITIALIZER**.

7.8.4.5 to_string() [3/6]

```
DDS_ReturnCode_t DDS_TopicQos::to_string (
            char * string,
             DDS_UnsignedLong & string_size,
            const  DDS_QosPrintFormat & format ) const  [inline]
```

Obtains a string representation of this **DDS_TopicQos** (p. 1397).

This overload behaves the same as **DDS_TopicQos::to_string** (p. 67) but allows the caller to specify the **DDS_Qos↩ PrintFormat** (p. 1285) which is used to format the output.

Parameters

string	<<***out***>> *(p. 433)* The char buffer that will be used to store the string representation of this **DDS_TopicQos** (p. 1397). If NULL, this function will return the required length of this buffer through the string_size parameter.
string_size	<<***inout***>> *(p. 433)* Cannot be NULL. The size of the supplied buffer. If the supplied buffer is NULL, or if the supplied buffer is not large enough, this value will be updated with the required length of the buffer.
format	<<***in***>> *(p. 433)* The **DDS_QosPrintFormat** (p. 1285) to be used to format the output string.

Returns

DDS_RETCODE_OK (p. 532) if no error was encountered.

See also

DDS_TopicQos::to_string(char*, DDS_UnsignedLong&, const DDS_TopicQos&, const DDS_QosPrint↩ Format&) const (p. 70)

References **DDS_RETCODE_ERROR**, and **DDS_RETCODE_OK**.

7.8.4.6 to_string() [4/6]

```
DDS_ReturnCode_t DDS_TopicQos::to_string (
            char * string,
             DDS_UnsignedLong & string_size,
            const  DDS_TopicQos & base,
            const  DDS_QosPrintFormat & format ) const  [inline]
```

Obtains a string representation of this **DDS_TopicQos** (p. 1397).

Only the differences between this **DDS_TopicQos** (p. 1397) and the **DDS_TopicQos** (p. 1397) supplied as the base are printed to the string.

If the supplied buffer is NULL, the required length of the string is returned via the string_size parameter.

If the supplied buffer is not NULL, the string representation of the **DDS_TopicQos** (p. 1397) is written to the buffer.

Parameters

string	*<<**out**>> (p. 433)* The char buffer that will be used to store the string representation of this **DDS_TopicQos** (p. 1397). If NULL, this function will return the required length of this buffer through the string_size parameter.
string_size	*<<**inout**>> (p. 433)* Cannot be NULL. The size of the supplied buffer. If the supplied buffer is NULL, or if the supplied buffer is not large enough, this value will be updated with the required length of the buffer.
base	*<<**in**>> (p. 433)* The **DDS_TopicQos** (p. 1397) to be used as the base QoS profile. Only the differences with respect to this base profile will be included in the output string.
format	*<<**in**>> (p. 433)* The **DDS_QosPrintFormat** (p. 1285) to be used to format the output string.

Returns

DDS_RETCODE_OK (p. 532) if no error was encountered.

7.8.4.7 to_string() [5/6]

```
DDS_ReturnCode_t DDS_TopicQos::to_string (
            char * string,
             DDS_UnsignedLong & string_size,
            const  DDS_QosPrintAll_t &  ) const  [inline]
```

Obtains a string representation of this **DDS_TopicQos** (p. 1397).

This overload behaves the same as **DDS_TopicQos::to_string** (p. 67) but prints the entire **DDS_TopicQos** (p. 1397) object. The only valid value for the argument of type **DDS_QosPrintAll_t** (p. 1285) is **DDS_QOS_PRINT_ALL** (p. 169).

For example
```
TopicQos qos;
char *str = NULL;
DDS_UnsignedLong str_size = 0;
qos.to_string(str, str_size, QOS_PRINT_ALL);
```

The resultant string is formatted according to the default value for **DDS_QosPrintFormat** (p. 1285).

Parameters

string	<<***out***>> *(p. 433)* The char buffer that will be used to store the string representation of this **DDS_TopicQos** (p. 1397). If NULL, this function will return the required length of this buffer through the string_size parameter.
string_size	<<***inout***>> *(p. 433)* Cannot be NULL. The size of the supplied buffer. If the supplied buffer is NULL, or if the supplied buffer is not large enough, this value will be updated with the required length of the buffer.

Returns

DDS_RETCODE_OK (p. 532) if no error was encountered.

See also

DDS_TopicQos::to_string(char*, DDS_UnsignedLong&, const DDS_TopicQos&, const DDS_QosPrint↩ Format&) const (p. 70)

References **DDS_QosPrintFormat_INITIALIZER**.

7.8.4.8 to_string() [6/6]

```
DDS_ReturnCode_t DDS_TopicQos::to_string (
            char * string,
             DDS_UnsignedLong & string_size,
            const  DDS_QosPrintAll_t & ,
            const  DDS_QosPrintFormat & format ) const  [inline]
```

Obtains a string representation of this **DDS_TopicQos** (p. 1397).

This overload behaves the same as the **DDS_TopicQos::to_string(char*, DDS_UnsignedLong&, const DDS_Qos↩ PrintAll_t&) const** (p. 70) overload. The only difference is that it allows the caller to specify the **DDS_QosPrintFormat** (p. 1285) which is used to format the resultant string.

Parameters

string	<<***out***>> *(*p. *433)* The char buffer that will be used to store the string representation of this **DDS_TopicQos** (p. 1397). If NULL, this function will return the required length of this buffer through the string_size parameter.
string_size	<<***inout***>> *(*p. *433)* Cannot be NULL. The size of the supplied buffer. If the supplied buffer is NULL, or if the supplied buffer is not large enough, this value will be updated with the required length of the buffer.
format	<<***in***>> *(*p. *433)* The **DDS_QosPrintFormat** (p. 1285) to be used to format the output string.

Returns

DDS_RETCODE_OK (p. 532) if no error was encountered.

See also

DDS_TopicQos::to_string(char*, DDS_UnsignedLong&, const DDS_TopicQos&, const DDS_QosPrint↩ Format&) const (p. 70)

7.8.5 Variable Documentation

7.8.5.1 DDS_PRINT_FORMAT_PROPERTY_DEFAULT

```
const struct DDS_PrintFormatProperty DDS_PRINT_FORMAT_PROPERTY_DEFAULT
```

Sentinel constant indicating default values for **DDS_PrintFormatProperty** (p. 1253).

Pass this object instead of your own **DDS_PrintFormatProperty** (p. 1253) object to use the default property values:

```
retCode = FooTypeSupport::data_to_string(

    sample,

    DDS_PRINT_FORMAT_PROPERTY_DEFAULT);
```

See also

DDS_PrintFormatProperty (p. 1253)

7.8.5.2 DDS_FILTERSAMPLEINFO_DEFAULT

```
const struct DDS_FilterSampleInfo DDS_FILTERSAMPLEINFO_DEFAULT [extern]
```

Initializer for new **DDS_FilterSampleInfo** (p. 1153).

All new **DDS_FilterSampleInfo** (p. 1153) instances allocated on the stack should be initialized to this value. No memory is allocated.

See also

DDS_FilterSampleInfo (p. 1153)

7.9 Zero Copy Transfer Over Shared Memory

*<<**extension**>> (p. 432)* Zero Copy transfer over shared memory

*<<**extension**>> (p. 432)* Zero Copy transfer over shared memory

Note

For a description of Zero Copy transfer over shared memory and its benefits, and a tutorial, see the "Sending Large Data" chapter in the **RTI Connext User's Manual**.

For buildable **code examples**, see `https://community.rti.com/kb/flatdata-and-zerocopy-examples`.

Zero Copy transfer over shared memory is available in the C API, in the Traditional C++ API, and in the Modern C++ API.

7.10 User Data Type Support

Defines generic classes and macros to support user data types.

Classes

- struct **DDS_TypeAllocationParams_t**

  Configures whether or not to allocate pointer and optional members.
- struct **DDS_TypeDeallocationParams_t**

  Configures whether to release or not pointer and optional members.
- struct **Foo**

  A representative user-defined data type.
- struct **DDS_InstanceHandleSeq**

  *Instantiates **FooSeq** (p. 1965) < **DDS_InstanceHandle_t** (p. 76) > .*
- class **FooTypeSupport**

  *<<**interface**>> (p. 433) <<**generic**>> (p. 433) User data type specific interface.*
- class **DDSTypeSupport**

  *<<**interface**>> (p. 433) An abstract marker interface that has to be specialized for each concrete user data type that will be used by the application.*

Macros

- #define **DDS_TYPESUPPORT_CPP**(TTypeSupport, TData)

  Declares the interface required to support a user data type.
- #define **DDS_DATAWRITER_CPP**(TDataWriter, TData)

  Declares the interface required to support a user data type specific data writer.
- #define **DDS_DATAREADER_W_DATA_CONSISTENCY_CHECK**(TDataReader, TDataSeq, TData)

  Declares the interface required to support a user data type-specific data reader.

Typedefs

- typedef DDS_HANDLE_TYPE_NATIVE **DDS_InstanceHandle_t**

  Type definition for an instance handle.

Functions

- **DDS_Boolean DDS_InstanceHandle_equals** (const **DDS_InstanceHandle_t** *self, const **DDS_InstanceHandle_t** *other)

  Compares this instance handle with another handle for equality.
- int **DDS_InstanceHandle_compare** (const **DDS_InstanceHandle_t** *self, const **DDS_InstanceHandle_t** *other)

  Compares this instance handle with another handle.
- void **DDS_InstanceHandle_copy** (**DDS_InstanceHandle_t** *self, const **DDS_InstanceHandle_t** *other)

  Copies this instance handle into another handle.
- **DDS_Boolean DDS_InstanceHandle_is_nil** (const **DDS_InstanceHandle_t** *self)

  *Compare this handle to **DDS_HANDLE_NIL** (p. 78).*

Variables

- const **DDS_InstanceHandle_t DDS_HANDLE_NIL**

  The NIL instance handle.

7.10.1 Detailed Description

Defines generic classes and macros to support user data types.

DDS specifies strongly typed interfaces to read and write user data. For each data class defined by the application, there is a number of specialised classes that are required to facilitate the type-safe interaction of the application with RTI Connext.

RTI Connext provides an automatic means to generate all these type-specific classes with the rtiddsgen utility. The complete set of automatic classes created for a hypothetical user data type named **Foo** (p. 1916) are shown below.

an application data type named **Foo** (p. 1916) "

The macros defined here declare the strongly typed APIs needed to support an arbitrary user defined data of type **Foo** (p. 1916).

See also

the Code Generator User's Manual

7.10.2 Macro Definition Documentation

7.10.2.1 DDS_TYPESUPPORT_CPP

```
#define DDS_TYPESUPPORT_CPP(
              TTypeSupport,
              TData )
```

Declares the interface required to support a user data type.

Defines:

FooTypeSupport *(*p. *1979)* TypeSupport of type **Foo** (p. 1916), i.e. **FooTypeSupport** (p. 1979)

Examples

HelloWorldSupport.cxx.

7.10.2.2 DDS_DATAWRITER_CPP

```
#define DDS_DATAWRITER_CPP(
              TDataWriter,
              TData )
```

Declares the interface required to support a user data type specific data writer.

Uses:

FooTypeSupport *(*p. *1979)* user data type, **Foo** (p. 1916)

Defines:

FooDataWriter *(*p. *1944)* **DDSDataWriter** (p. 1589) of type **Foo** (p. 1916), i.e. **FooDataWriter** (p. 1944)

7.10.2.3 DDS_DATAREADER_W_DATA_CONSISTENCY_CHECK

```
#define DDS_DATAREADER_W_DATA_CONSISTENCY_CHECK(
             TDataReader,
             TDataSeq,
             TData )
```

Declares the interface required to support a user data type-specific data reader.

Uses:

FooTypeSupport *(*p. *1979)* user data type, **Foo** (p. 1916)
FooSeq (p. 1965) sequence of user data type, sequence<::Foo>

Defines:

FooDataReader *(*p. *1917)* **DDSDataReader** (p. 1556) of type **Foo** (p. 1916), i.e. **FooDataReader** (p. 1917)

See also

FooSeq (p. 1965)

Examples

HelloWorldSupport.cxx.

7.10.3 Typedef Documentation

7.10.3.1 DDS_InstanceHandle_t

```
typedef DDS_HANDLE_TYPE_NATIVE  DDS_InstanceHandle_t
```

Type definition for an instance handle.

Handle to identify different instances of the same **DDSTopic** (p. 1885) of a certain type.

See also

FooDataWriter::register_instance (p. 1946)
DDS_SampleInfo::instance_handle (p. 1349)

7.10.4 Function Documentation

7.10.4.1 DDS_InstanceHandle_equals()

```
DDS_Boolean DDS_InstanceHandle_equals (
            const  DDS_InstanceHandle_t * self,
            const  DDS_InstanceHandle_t * other )
```

Compares this instance handle with another handle for equality.

Parameters

self	*<<**in**>> (p. 433)* This handle. Cannot be NULL.
other	*<<**in**>> (p. 433)* The other handle to be compared with this handle. Cannot be NULL.

Returns

DDS_BOOLEAN_TRUE (p. 513) if the two handles have equal values, or **DDS_BOOLEAN_FALSE** (p. 513) otherwise.

See also

DDS_InstanceHandle_is_nil (p. 78)

7.10.4.2 DDS_InstanceHandle_compare()

```
int DDS_InstanceHandle_compare (
             const  DDS_InstanceHandle_t * self,
             const  DDS_InstanceHandle_t * other )
```

Compares this instance handle with another handle.

Parameters

self	*<<**in**>> (p. 433)* This handle. Cannot be NULL.
other	*<<**in**>> (p. 433)* The other handle to be compared with this handle. Cannot be NULL.

Returns

If the two handles are equal, the function returns 0. If self is greater than other the function returns a positive number; otherwise, it returns a negative number.

See also

DDS_InstanceHandle_is_nil (p. 78)

7.10.4.3 DDS_InstanceHandle_copy()

```
void DDS_InstanceHandle_copy (
              DDS_InstanceHandle_t * self,
             const  DDS_InstanceHandle_t * other )
```

Copies this instance handle into another handle.

Parameters

self	<<***in***>> *(p. 433)* The source handle. Cannot be NULL.
other	<<***in***>> *(p. 433)* The destination handle to be copied into. Cannot be NULL.

7.10.4.4 DDS_InstanceHandle_is_nil()

```
DDS_Boolean DDS_InstanceHandle_is_nil (
            const  DDS_InstanceHandle_t * self )
```

Compare this handle to **DDS_HANDLE_NIL** (p. 78).

Returns

DDS_BOOLEAN_TRUE (p. 513) if the given instance handle is equal to **DDS_HANDLE_NIL** (p. 78) or **DDS_↩ BOOLEAN_FALSE** (p. 513) otherwise.

See also

DDS_InstanceHandle_equals (p. 76)

7.10.5 Variable Documentation

7.10.5.1 DDS_HANDLE_NIL

```
const  DDS_InstanceHandle_t DDS_HANDLE_NIL  [extern]
```

The NIL instance handle.

Special **DDS_InstanceHandle_t** (p. 76) value

See also

DDS_InstanceHandle_is_nil (p. 78)

Examples

HelloWorld_publisher.cxx.

7.11 Type Code Support

*<<**extension**>> (p. 432)* A ***DDS_TypeCode*** *(p. 1428)* is a mechanism for representing a type at runtime. RTI Connext can use type codes to send type definitions on the network. You will need to understand this API in order to use the **Dynamic Data** (p. 103) capability or to inspect the type information you receive from remote readers and writers.

Classes

- struct **DDS_TypeCodePrintFormatProperty**

  A collection of attributes used to configure how a TypeCode appears when converted to a string.
- struct **DDS_TypeCode**

  The definition of a particular data type, which you can use to inspect the name, members, and other properties of types generated with rtiddsgen (see the `Code Generator User's Manual`) or to modify types you define yourself at runtime.
- struct **DDS_StructMember**

  A description of a member of a struct.
- struct **DDS_StructMemberSeq**

  Defines a sequence of struct members.
- struct **DDS_UnionMember**

  A description of a member of a union.
- struct **DDS_UnionMemberSeq**

  Defines a sequence of union members.
- struct **DDS_EnumMember**

  A description of a member of an enumeration.
- struct **DDS_EnumMemberSeq**

  Defines a sequence of enumerator members.
- struct **DDS_ValueMember**

  A description of a member of a value type.
- struct **DDS_ValueMemberSeq**

  Defines a sequence of value members.
- struct **DDS_TypeCodeFactory**

  A singleton factory for creating, copying, and deleting data type definitions dynamically.

Macros

- #define **DDS_TYPECODE_MEMBER_ID_INVALID**

  *A sentinel indicating an invalid **DDS_TypeCode** (p. 1428) member ID.*
- #define **DDS_TYPECODE_INDEX_INVALID**

  *A sentinel indicating an invalid **DDS_TypeCode** (p. 1428) member index.*
- #define **DDS_TYPECODE_NOT_BITFIELD**

  Indicates that a member of a type is not a bitfield.
- #define **DDS_VM_NONE**

  Constant used to indicate that a value type has no modifiers.
- #define **DDS_VM_CUSTOM**

  Constant used to indicate that a value type has the `custom` modifier.
- #define **DDS_VM_ABSTRACT**

Constant used to indicate that a value type has the `abstract` modifier.

- #define **DDS_VM_TRUNCATABLE**

  Constant used to indicate that a value type has the `truncatable` modifier.

- #define **DDS_PRIVATE_MEMBER**

  Constant used to indicate that a value type member is private.

- #define **DDS_PUBLIC_MEMBER**

  Constant used to indicate that a value type member is public.

- #define **DDS_TYPECODE_NONKEY_MEMBER**

  A flag indicating that a type member is optional and not part of the key.

- #define **DDS_TYPECODE_KEY_MEMBER**

  A flag indicating that a type member is part of the key for that type, and therefore required.

- #define **DDS_TYPECODE_NONKEY_REQUIRED_MEMBER**

  A flag indicating that a type member is not part of the key but is nevertheless required.

- #define **DDS_TypeCode_PrintFormat_INITIALIZER**

  *Static initializer for **DDS_TypeCodePrintFormatProperty** (p. 1490).*

Typedefs

- typedef short **DDS_ValueModifier**

  Modifier type for a value type.

- typedef short **DDS_Visibility**

  Type to indicate the visibility of a value type member.

Enumerations

- enum **DDS_TypeCodePrintFormatKind** {
  DDS_TYPE_CODE_PRINT_KIND_IDL ,
  DDS_TYPE_CODE_PRINT_KIND_XML ,
  DDS_TYPE_CODE_PRINT_KIND_JSON_SCHEMA }

  *The different formats to print a **DDS_TypeCode** (p. 1428).*

- enum **DDS_TCKind** {
  DDS_TK_NULL ,
  DDS_TK_SHORT ,
  DDS_TK_LONG ,
  DDS_TK_USHORT ,
  DDS_TK_ULONG ,
  DDS_TK_FLOAT ,
  DDS_TK_DOUBLE ,
  DDS_TK_BOOLEAN ,
  DDS_TK_CHAR ,
  DDS_TK_OCTET ,
  DDS_TK_STRUCT ,
  DDS_TK_UNION ,
  DDS_TK_ENUM ,
  DDS_TK_STRING ,
  DDS_TK_SEQUENCE ,
  DDS_TK_ARRAY ,
  DDS_TK_ALIAS ,

DDS_TK_LONGLONG ,
DDS_TK_ULONGLONG ,
DDS_TK_LONGDOUBLE ,
DDS_TK_WCHAR ,
DDS_TK_WSTRING ,
DDS_TK_VALUE ,
DDS_TK_INT8 ,
DDS_TK_UINT8 ,
DDS_TK_RAW_BYTES = 0x7e ,
DDS_TK_RAW_BYTES_KEYED = 0x7f ,
DDS_TK_FINAL_EXTENSIBILITY = 0x4000 ,
DDS_TK_MUTABLE_EXTENSIBILITY = 0x2000 ,
DDS_TK_FLAT_DATA_LANGUAGE_BINDING = RTI_XCDR_TK_FLAGS_IS_FLAT_DATA ,
DDS_TK_SHMEM_REF_TRANSFER_MODE = RTI_XCDR_TK_FLAGS_IS_SHMEM_REF }

*Enumeration type for **DDS_TypeCode** (p. 1428) kinds.*

- enum **DDS_ExtensibilityKind** {
DDS_FINAL_EXTENSIBILITY ,
DDS_EXTENSIBLE_EXTENSIBILITY ,
DDS_MUTABLE_EXTENSIBILITY }

  Type to indicate the extensibility of a type.

Variables

- **DDS_TypeCode DDS_g_tc_null**

  Basic NULL type.
- **DDS_TypeCode DDS_g_tc_short**

  Basic 16-bit signed integer type.
- **DDS_TypeCode DDS_g_tc_long**

  Basic 32-bit signed integer type.
- **DDS_TypeCode DDS_g_tc_ushort**

  Basic unsigned 16-bit integer type.
- **DDS_TypeCode DDS_g_tc_ulong**

  Basic unsigned 32-bit integer type.
- **DDS_TypeCode DDS_g_tc_float**

  Basic 32-bit floating point type.
- **DDS_TypeCode DDS_g_tc_double**

  Basic 64-bit floating point type.
- **DDS_TypeCode DDS_g_tc_boolean**

  Basic Boolean type.
- **DDS_TypeCode DDS_g_tc_char**

  Basic single-byte character type.
- **DDS_TypeCode DDS_g_tc_octet**

  Basic octet/byte type.
- **DDS_TypeCode DDS_g_tc_int8**

  Basic signed 8-bit int type.
- **DDS_TypeCode DDS_g_tc_uint8**

  Basic unsigned 8-bit int type.
- **DDS_TypeCode DDS_g_tc_longlong**

Basic 64-bit integer type.

- **DDS_TypeCode DDS_g_tc_ulonglong**

  Basic unsigned 64-bit integer type.

- **DDS_TypeCode DDS_g_tc_longdouble**

  Basic 128-bit floating point type.

- **DDS_TypeCode DDS_g_tc_wchar**

  Basic four-byte character type.

7.11.1 Detailed Description

<<***extension***>> *(p. 432)* A ***DDS_TypeCode*** *(p. 1428)* is a mechanism for representing a type at runtime. RTI Connext can use type codes to send type definitions on the network. You will need to understand this API in order to use the **Dynamic Data** (p. 103) capability or to inspect the type information you receive from remote readers and writers.

Type codes are values that are used to describe arbitrarily complex types at runtime. Type code values are manipulated via the **DDS_TypeCode** (p. 1428) class, which has an analogue in CORBA.

A **DDS_TypeCode** (p. 1428) value consists of a type code *kind* (represented by the **DDS_TCKind** (p. 89) enumeration) and a list of *members* (that is, fields). These members are recursive: each one has its own **DDS_TypeCode** (p. 1428), and in the case of complex types (structures, arrays, and so on), these contained type codes contain their own members.

There are a number of uses for type codes. The type code mechanism can be used to unambiguously match type representations. The **DDS_TypeCode::equal** (p. 1435) method is a more reliable test than comparing the string type names, requiring equivalent definitions of the types.

7.11.2 Accessing a Local ::DDS_TypeCode

When generating types with rtiddsgen, type codes are always enabled. For these types, a **DDS_TypeCode** (p. 1428) may be accessed by calling the `Foo_get_typecode` function for a type "Foo", which returns a **DDS_TypeCode** (p. 1428) pointer. via the **FooTypeCode.VALUE** member. This API also includes support for dynamic creation of **DDS_TypeCode** (p. 1428) values, typically for use with the **Dynamic Data** (p. 103) API. You can create a **DDS_Type↩ Code** (p. 1428) using the **DDS_TypeCodeFactory** (p. 1476) class. You will construct the **DDS_TypeCode** (p. 1428) recursively, from the outside in: start with the type codes for primitive types, then compose them into complex types like arrays, structures, and so on. You will find the following methods helpful:

- **DDS_TypeCodeFactory::get_primitive_tc** (p. 1480), which provides the **DDS_TypeCode** (p. 1428) instances corresponding to the primitive types (e.g. **DDS_TK_LONG** (p. 89), **DDS_TK_SHORT** (p. 89), and so on).
- **DDS_TypeCodeFactory::create_string_tc** (p. 1487) and **DDS_TypeCodeFactory::create_wstring_↩ tc** (p. 1487) create a **DDS_TypeCode** (p. 1428) representing a text string with a certain *bound* (i.e. maximum length).
- **DDS_TypeCodeFactory::create_array_tc** (p. 1489) and **DDS_TypeCodeFactory::create_sequence_↩ tc** (p. 1488) create a **DDS_TypeCode** (p. 1428) for a collection based on the **DDS_TypeCode** (p. 1428) for its elements.
- **DDS_TypeCodeFactory::create_struct_tc** (p. 1480) and **DDS_TypeCodeFactory::create_value_tc** (p. 1482) create a **DDS_TypeCode** (p. 1428) for a structured type.

7.11.3 Accessing a Remote ::DDS_TypeCode

In addition to being used locally, RTI Connext can transmit **DDS_TypeCode** (p. 1428) on the network between participants. This information can be used to access information about types used remotely at runtime, for example to be able to publish or subscribe to topics of arbitrarily types (see **Dynamic Data** (p. 103)). This functionality is useful for a generic system monitoring tool like `rtiddsspy`.

Remote **DDS_TypeCode** (p. 1428) information is shared during discovery over the publication and subscription built-in topics and can be accessed using the built-in readers for these topics; see **Built-in Topics** (p. 61). Discovered **DDS_TypeCode** (p. 1428) values are not cached by RTI Connext upon receipt and are therefore not available from the built-in topic data returned by **DDSDataWriter::get_matched_subscription_data** (p. 1600) or **DDSDataReader::get↩ _matched_publication_data** (p. 1568).

The space available locally to deserialize a discovered remote **DDS_TypeCode** (p. 1428) is specified by the **DDSDomainParticipant** (p. 1619)'s **DDS_DomainParticipantResourceLimitsQosPolicy::type_code_max_↩ serialized_length** (p. 994) QoS parameter. To support especially complex type codes, it may be necessary for you to increase the value of this parameter.

See also

DDS_TypeCode (p. 1428)

Dynamic Data (p. 103)

the `Code Generator User's Manual`

DDS_SubscriptionBuiltinTopicData (p. 1370)

DDS_PublicationBuiltinTopicData (p. 1264)

7.11.4 Macro Definition Documentation

7.11.4.1 DDS_TYPECODE_MEMBER_ID_INVALID

```
#define DDS_TYPECODE_MEMBER_ID_INVALID
```

A sentinel indicating an invalid **DDS_TypeCode** (p. 1428) member ID.

7.11.4.2 DDS_TYPECODE_INDEX_INVALID

```
#define DDS_TYPECODE_INDEX_INVALID
```

A sentinel indicating an invalid **DDS_TypeCode** (p. 1428) member index.

7.11.4.3 DDS_TYPECODE_NOT_BITFIELD

```
#define DDS_TYPECODE_NOT_BITFIELD
```

Indicates that a member of a type is not a bitfield.

7.11.4.4 DDS_VM_NONE

```
#define DDS_VM_NONE
```

Constant used to indicate that a value type has no modifiers.

See also

DDS_ValueModifier (p. 87)

Examples

HelloWorld.cxx.

7.11.4.5 DDS_VM_CUSTOM

```
#define DDS_VM_CUSTOM
```

Constant used to indicate that a value type has the `custom` modifier.

This modifier is used to specify whether the value type uses custom marshaling.

See also

DDS_ValueModifier (p. 87)

7.11.4.6 DDS_VM_ABSTRACT

```
#define DDS_VM_ABSTRACT
```

Constant used to indicate that a value type has the `abstract` modifier.

An abstract value type may not be instantiated.

See also

DDS_ValueModifier (p. 87)

7.11.4.7 DDS_VM_TRUNCATABLE

```
#define DDS_VM_TRUNCATABLE
```

Constant used to indicate that a value type has the `truncatable` modifier.

A value with a state that derives from another value with a state can be declared as truncatable. A truncatable type means the object can be truncated to the base type.

See also

DDS_ValueModifier (p. 87)

7.11.4.8 DDS_PRIVATE_MEMBER

```
#define DDS_PRIVATE_MEMBER
```

Constant used to indicate that a value type member is private.

See also

DDS_Visibility (p. 88)
DDS_PUBLIC_MEMBER (p. 85)

7.11.4.9 DDS_PUBLIC_MEMBER

```
#define DDS_PUBLIC_MEMBER
```

Constant used to indicate that a value type member is public.

See also

DDS_Visibility (p. 88)
DDS_PRIVATE_MEMBER (p. 85)

Examples

HelloWorld.cxx.

7.11.4.10 DDS_TYPECODE_NONKEY_MEMBER

```
#define DDS_TYPECODE_NONKEY_MEMBER
```

A flag indicating that a type member is optional and not part of the key.

If a type is used with the **Dynamic Data** (p. 103) facility, a **DDS_DynamicData** (p. 1013) sample of the type will only contain a value for a **DDS_TYPECODE_NONKEY_MEMBER** (p. 85) field if one has been explicitly set (see, for example, **DDS_DynamicData::set_long** (p. 1093)). The middleware will *not* assume any default value.

See also

DDS_TYPECODE_KEY_MEMBER (p. 86)
DDS_TYPECODE_NONKEY_REQUIRED_MEMBER (p. 86)
DDS_TYPECODE_KEY_MEMBER (p. 86)
DDS_TypeCode::add_member (p. 1463)
DDS_TypeCode::add_member_ex (p. 1465)
DDS_TypeCode::is_member_key (p. 1443)
DDS_TypeCode::is_member_required (p. 1443)
DDS_StructMember::is_key (p. 1366)
DDS_ValueMember::is_key (p. 1507)

7.11.4.11 DDS_TYPECODE_KEY_MEMBER

```
#define DDS_TYPECODE_KEY_MEMBER
```

A flag indicating that a type member is part of the key for that type, and therefore required.

If a type is used with the **Dynamic Data** (p. 103) facility, all **DDS_DynamicData** (p. 1013) samples of the type will contain a value for all **DDS_TYPECODE_KEY_MEMBER** (p. 86) fields. If you do not set a value of the member explicitly (see, for example, **DDS_DynamicData::set_long** (p. 1093)), the middleware will assume a default "zero" value: numeric values will be set to zero; strings and sequences will be of zero length.

See also

DDS_TYPECODE_NONKEY_REQUIRED_MEMBER (p. 86)
DDS_TYPECODE_NONKEY_MEMBER (p. 85)
DDS_TypeCode::add_member (p. 1463)
DDS_TypeCode::add_member_ex (p. 1465)
DDS_TypeCode::is_member_key (p. 1443)
DDS_TypeCode::is_member_required (p. 1443)
DDS_StructMember::is_key (p. 1366)
DDS_ValueMember::is_key (p. 1507)

7.11.4.12 DDS_TYPECODE_NONKEY_REQUIRED_MEMBER

```
#define DDS_TYPECODE_NONKEY_REQUIRED_MEMBER
```

A flag indicating that a type member is not part of the key but is nevertheless required.

This is the most common kind of member.

If a type is used with the **Dynamic Data** (p. 103) facility, all **DDS_DynamicData** (p. 1013) samples of the type will contain a value for all **DDS_TYPECODE_NONKEY_REQUIRED_MEMBER** (p. 86) fields. If you do not set a value of the member explicitly (see, for example, **DDS_DynamicData::set_long** (p. 1093)), the middleware will assume a default "zero" value: numeric values will be set to zero; strings and sequences will be of zero length.

See also

DDS_TYPECODE_KEY_MEMBER (p. 86)
DDS_TYPECODE_NONKEY_MEMBER (p. 85)
DDS_TYPECODE_KEY_MEMBER (p. 86)
DDS_TypeCode::add_member (p. 1463)
DDS_TypeCode::add_member_ex (p. 1465)
DDS_TypeCode::is_member_key (p. 1443)
DDS_TypeCode::is_member_required (p. 1443)
DDS_StructMember::is_key (p. 1366)
DDS_ValueMember::is_key (p. 1507)

7.11.4.13 DDS_TypeCode_PrintFormat_INITIALIZER

```
#define DDS_TypeCode_PrintFormat_INITIALIZER
```

Static initializer for **DDS_TypeCodePrintFormatProperty** (p. 1490).

Use this initializer to ensure that new objects don't have uninitialized contents.

```
struct DDS_TypeCodePrintFormatProperty property = DDS_TypeCodePrintFormatProperty;
```

7.11.5 Typedef Documentation

7.11.5.1 DDS_ValueModifier

```
typedef short DDS_ValueModifier
```

Modifier type for a value type.

See also

DDS_VM_NONE (p. 84)
DDS_VM_CUSTOM (p. 84)
DDS_VM_ABSTRACT (p. 84)
DDS_VM_TRUNCATABLE (p. 84)

7.11.5.2 DDS_Visibility

```
typedef short DDS_Visibility
```

Type to indicate the visibility of a value type member.

See also

DDS_PRIVATE_MEMBER (p. 85)
DDS_PUBLIC_MEMBER (p. 85)

7.11.6 Enumeration Type Documentation

7.11.6.1 DDS_TypeCodePrintFormatKind

```
enum DDS_TypeCodePrintFormatKind
```

The different formats to print a **DDS_TypeCode** (p. 1428).

Enumerator

DDS_TYPE_CODE_PRINT_KIND_IDL	Indicates that a **DDS_TypeCode** (p. 1428) is to be printed in IDL format.
DDS_TYPE_CODE_PRINT_KIND_XML	Indicates that a **DDS_TypeCode** (p. 1428) is to be printed in XML format.
DDS_TYPE_CODE_PRINT_KIND_JSON_SCHEMA	Indicates that a **DDS_TypeCode** (p. 1428) is printed in JSON Schema format.

7.11.6.2 DDS_TCKind

enum **DDS_TCKind**

Enumeration type for **DDS_TypeCode** (p. 1428) kinds.

Type code kinds are modeled as values of this type.

Enumerator

DDS_TK_NULL	Indicates that a type code does not describe anything.
DDS_TK_SHORT	short type.
DDS_TK_LONG	long type.
DDS_TK_USHORT	unsigned short type.
DDS_TK_ULONG	unsigned long type.
DDS_TK_FLOAT	float type.
DDS_TK_DOUBLE	double type.
DDS_TK_BOOLEAN	boolean type.
DDS_TK_CHAR	char type.
DDS_TK_OCTET	octet type.
DDS_TK_STRUCT	struct type.
DDS_TK_UNION	union type.
DDS_TK_ENUM	enumerated type.
DDS_TK_STRING	string type.
DDS_TK_SEQUENCE	sequence type.
DDS_TK_ARRAY	array type.
DDS_TK_ALIAS	alias (typedef) type.
DDS_TK_LONGLONG	long long type.
DDS_TK_ULONGLONG	unsigned long long type.
DDS_TK_LONGDOUBLE	long double type.
DDS_TK_WCHAR	wide char type.
DDS_TK_WSTRING	wide string type.
DDS_TK_VALUE	value type.
DDS_TK_INT8	8-bit signed integer type.
DDS_TK_UINT8	8-bit unsigned integer type.

7.11.6.3 DDS_ExtensibilityKind

enum **DDS_ExtensibilityKind**

Type to indicate the extensibility of a type.

Enumerator

DDS_FINAL_EXTENSIBILITY	Specifies that a type has FINAL extensibility. A type may be final, indicating that the range of its possible values is strictly defined. In particular, it is not possible to add elements to members of collection or aggregated types while maintaining type assignability. The following types are always final: • **DDS_TK_ARRAY** (p. 89) • All primitive types
DDS_EXTENSIBLE_EXTENSIBILITY	Specifies that a type has EXTENSIBLE extensibility. A type may be appendable (formerly called extensible), indicating that two types, where one contains all of the elements/members of the other plus additional elements/members appended to the end, may remain assignable.
DDS_MUTABLE_EXTENSIBILITY	Specifies that a type has MUTABLE extensibility. A type may be mutable, indicating that two types may differ from one another in the additional, removal, and/or transposition of elements/members while remaining assignable. The following types are always mutable: • **DDS_TK_SEQUENCE** (p. 89) • **DDS_TK_STRING** (p. 89) • **DDS_TK_WSTRING** (p. 89)

7.11.7 Variable Documentation

7.11.7.1 DDS_g_tc_null

```
DDS_TypeCode DDS_g_tc_null
```

Basic NULL type.

For new code, **DDS_TypeCodeFactory::get_primitive_tc** (p. 1480) is preferred to using this global variable.

See also

DDS_TypeCodeFactory::get_primitive_tc (p. 1480)

7.11.7.2 DDS_g_tc_short

```
DDS_TypeCode DDS_g_tc_short
```

Basic 16-bit signed integer type.

For new code, **DDS_TypeCodeFactory::get_primitive_tc** (p. 1480) is preferred to using this global variable.

See also

DDS_TypeCodeFactory::get_primitive_tc (p. 1480)

DDS_Short (p. 514)

7.11.7.3 DDS_g_tc_long

```
DDS_TypeCode DDS_g_tc_long
```

Basic 32-bit signed integer type.

For new code, **DDS_TypeCodeFactory::get_primitive_tc** (p. 1480) is preferred to using this global variable.

See also

DDS_TypeCodeFactory::get_primitive_tc (p. 1480)

DDS_Long (p. 515)

7.11.7.4 DDS_g_tc_ushort

```
DDS_TypeCode DDS_g_tc_ushort
```

Basic unsigned 16-bit integer type.

For new code, **DDS_TypeCodeFactory::get_primitive_tc** (p. 1480) is preferred to using this global variable.

See also

DDS_TypeCodeFactory::get_primitive_tc (p. 1480)

DDS_UnsignedShort (p. 514)

7.11.7.5 DDS_g_tc_ulong

```
DDS_TypeCode DDS_g_tc_ulong
```

Basic unsigned 32-bit integer type.

For new code, **DDS_TypeCodeFactory::get_primitive_tc** (p. 1480) is preferred to using this global variable.

See also

DDS_TypeCodeFactory::get_primitive_tc (p. 1480)
DDS_UnsignedLong (p. 515)

7.11.7.6 DDS_g_tc_float

```
DDS_TypeCode DDS_g_tc_float
```

Basic 32-bit floating point type.

For new code, **DDS_TypeCodeFactory::get_primitive_tc** (p. 1480) is preferred to using this global variable.

See also

DDS_TypeCodeFactory::get_primitive_tc (p. 1480)
DDS_Float (p. 515)

7.11.7.7 DDS_g_tc_double

```
DDS_TypeCode DDS_g_tc_double
```

Basic 64-bit floating point type.

For new code, **DDS_TypeCodeFactory::get_primitive_tc** (p. 1480) is preferred to using this global variable.

See also

DDS_TypeCodeFactory::get_primitive_tc (p. 1480)
DDS_Double (p. 515)

7.11.7.8 DDS_g_tc_boolean

```
DDS_TypeCode DDS_g_tc_boolean
```

Basic Boolean type.

For new code, **DDS_TypeCodeFactory::get_primitive_tc** (p. 1480) is preferred to using this global variable.

See also

DDS_TypeCodeFactory::get_primitive_tc (p. 1480)

DDS_Boolean (p. 516)

7.11.7.9 DDS_g_tc_char

```
DDS_TypeCode DDS_g_tc_char
```

Basic single-byte character type.

For new code, **DDS_TypeCodeFactory::get_primitive_tc** (p. 1480) is preferred to using this global variable.

See also

DDS_TypeCodeFactory::get_primitive_tc (p. 1480)

DDS_Char (p. 513)

7.11.7.10 DDS_g_tc_octet

```
DDS_TypeCode DDS_g_tc_octet
```

Basic octet/byte type.

For new code, **DDS_TypeCodeFactory::get_primitive_tc** (p. 1480) is preferred to using this global variable.

See also

DDS_TypeCodeFactory::get_primitive_tc (p. 1480)

DDS_Octet (p. 514)

7.11.7.11 DDS_g_tc_int8

```
DDS_TypeCode DDS_g_tc_int8
```

Basic signed 8-bit int type.

For new code, **DDS_TypeCodeFactory::get_primitive_tc** (p. 1480) is preferred to using this global variable.

See also

DDS_TypeCodeFactory::get_primitive_tc (p. 1480)
DDS_Int8 (p. 514)

7.11.7.12 DDS_g_tc_uint8

```
DDS_TypeCode DDS_g_tc_uint8
```

Basic unsigned 8-bit int type.

For new code, **DDS_TypeCodeFactory::get_primitive_tc** (p. 1480) is preferred to using this global variable.

See also

DDS_TypeCodeFactory::get_primitive_tc (p. 1480)
DDS_UInt8 (p. 514)

7.11.7.13 DDS_g_tc_longlong

```
DDS_TypeCode DDS_g_tc_longlong
```

Basic 64-bit integer type.

For new code, **DDS_TypeCodeFactory::get_primitive_tc** (p. 1480) is preferred to using this global variable.

See also

DDS_TypeCodeFactory::get_primitive_tc (p. 1480)
DDS_LongLong (p. 515)

7.11.7.14 DDS_g_tc_ulonglong

```
DDS_TypeCode DDS_g_tc_ulonglong
```

Basic unsigned 64-bit integer type.

For new code, **DDS_TypeCodeFactory::get_primitive_tc** (p. 1480) is preferred to using this global variable.

See also

DDS_TypeCodeFactory::get_primitive_tc (p. 1480)
DDS_UnsignedLongLong (p. 515)

7.11.7.15 DDS_g_tc_longdouble

```
DDS_TypeCode DDS_g_tc_longdouble
```

Basic 128-bit floating point type.

For new code, **DDS_TypeCodeFactory::get_primitive_tc** (p. 1480) is preferred to using this global variable.

See also

DDS_TypeCodeFactory::get_primitive_tc (p. 1480)
DDS_LongDouble (p. 516)

7.11.7.16 DDS_g_tc_wchar

```
DDS_TypeCode DDS_g_tc_wchar
```

Basic four-byte character type.

For new code, **DDS_TypeCodeFactory::get_primitive_tc** (p. 1480) is preferred to using this global variable.

See also

DDS_TypeCodeFactory::get_primitive_tc (p. 1480)
DDS_Wchar (p. 514)

7.12 Built-in Types

RTI Connext provides a set of very simple data types for you to use with the topics in your application.

Modules

- **String Built-in Type**

  Built-in type consisting of a single character string.
- **KeyedString Built-in Type**

  Built-in type consisting of a string payload and a second string that is the key.
- **Octets Built-in Type**

  Built-in type consisting of a variable-length array of opaque bytes.
- **KeyedOctets Built-in Type**

  Built-in type consisting of a variable-length array of opaque bytes and a string that is the key.

7.12.1 Detailed Description

RTI Connext provides a set of very simple data types for you to use with the topics in your application.

The middleware provides four built-in types:

- String: A payload consisting of a single string of characters. This type has no key.
- **DDS_KeyedString** (p. 1171): A payload consisting of a single string of characters and a second string, the key, that identifies the instance to which the sample belongs.
- **DDS_Octets** (p. 1217): A payload consisting of an opaque variable-length array of bytes. This type has no key.
- **DDS_KeyedOctets** (p. 1168): A payload consisting of an opaque variable-length array of bytes and a string, the key, that identifies the instance to which the sample belongs.

The String and **DDS_KeyedString** (p. 1171) types are appropriate for simple text-based applications. The **DDS_Octets** (p. 1217) and **DDS_KeyedOctets** (p. 1168) types are appropriate for applications that perform their own custom data serialization, such as legacy applications still in the process of migrating to RTI Connext. In most cases, string-based or structured data is preferable to opaque data, because the latter cannot be easily visualized in tools or used with content-based filters (see **DDSContentFilteredTopic** (p. 1551)).

The built-in types are very simple in order to get you up and running as quickly as possible. If you need a structured data type you can define your own type with exactly the fields you need in one of two ways:

- At compile time, by generating code from an IDL or XML file using the rtiddsgen utility
- At runtime, by using the **Dynamic Data** (p. 103) API

7.12.2 Managing Memory for Builtin Types

When a sample is written, the DataWriter serializes it and stores the result in a buffer obtained from a pool of preallocated buffers. In the same way, when a sample is received, the DataReader deserializes it and stores the result in a sample coming from a pool of preallocated samples.

For builtin types, the maximum size of the buffers/samples and depends on the nature of the application using the builtin type.

You can configure the maximum size of the builtin types on a per-DataWriter and per-DataReader basis using the **DDS_PropertyQosPolicy** (p. 1260) in DataWriters, DataReaders or Participants.

The following table lists the supported builtin type properties to configure memory allocation. When the properties are defined in the DomainParticipant, they are applicable to all DataWriters and DataReaders belonging to the Domain↩ Participant unless they are overwrittem in the DataWriters and DataReaders.

Table 7.33 ***Builtin Types Allocation Properties***

Property	Description
dds.builtin_type.string.alloc_size	Maximum size of the strings published by the **DDSStringDataWriter** (p. 1852) or received the **DDSStringDataReader** (p. 1848) (includes the NULL-terminated character). Default: dds.builtin_type.string.max_size if defined. Otherwise, 1024.
dds.builtin_type.keyed_string.alloc_key_size	Maximum size of the keys used by the **DDSKeyedStringDataWriter** (p. 1778) or **DDSKeyedStringDataReader** (p. 1769) (includes the NULL-terminated character). Default: dds.builtin_type.keyed_string.max_key_size if defined. Otherwise, 1024.
dds.builtin_type.keyed_string.alloc_size	Maximum size of the strings published by the **DDSKeyedString↩ DataWriter** (p. 1778) or received by the **DDSKeyedStringData↩ Reader** (p. 1769) (includes the NULL-terminated character). Default: dds.builtin_type.keyed_string.max_size if defined. Otherwise, 1024.
dds.builtin_type.octets.alloc_size	Maximum size of the octet sequences published the **DDSOctets↩ DataWriter** (p. 1804) or received by the **DDSOctetsDataReader** (p. 1800). Default: dds.builtin_type.octets.max_size if defined. Otherwise, 2048.
dds.builtin_type.keyed_octets.alloc_key_size	Maximum size of the key published by the **DDSKeyedOctetsData↩ Writer** (p. 1750) or received by the **DDSKeyedOctetsDataReader** (p. 1741) (includes the NULL-terminated character). Default: dds.builtin_type.keyed_octets.max_key_size if defined. Otherwise, 1024.
dds.builtin_type.keyed_octets.alloc_size	Maximum size of the octets sequences published by a **DDSKeyed↩ OctetsDataWriter** (p. 1750) or received by a **DDSKeyedOctets↩ DataReader** (p. 1741). Default: dds.builtin_type.keyed_octets.max_size if defined. Otherwise, 2048.

The previous properties must be set consistently with respect to the corresponding $*$.max_size properties that set the maximum size of the builtin types in the typecode.

7.12.3 Typecodes for Builtin Types

The typecodes associated with the builtin types are generated from the following IDL type definitions:

```
module DDS {

    struct String {

        string value;

    };


    struct KeyedString {

        string key;
```

```
        string value;
    };

    struct Octets {
        sequence<octet> value;
    };

    struct KeyedOctets {
        string key;
        sequence<octet> value;
    };
};
```

The maximum size of the strings and sequences that will be included in the type code definitions can be configured on a per-DomainParticipant-basis by using the properties in following table.

Table 7.34 ***Properties for Allocating Size of Builtin Types, per DomainParticipant***

Property	Description
dds.builtin_type.string.max_size	Maximum size of the strings published by the StringDataWriters and received by the StringDataReaders belonging to a DomainParticipant (includes the NULL-terminated character). Default: 1024.
dds.builtin_type.keyed_string.max_key_size	Maximum size of the keys used by the KeyedStringDataWriters and KeyedStringDataReaders belonging to a DomainParticipant (includes the NULL-terminated character). Default: 1024.
dds.builtin_type.keyed_string.max_size	Maximum size of the strings published by the KeyedStringDataWriters and received by the KeyedStringDataReaders belonging to a DomainParticipant using the builtin type (includes the NULL-terminated character). Default: 1024
dds.builtin_type.octets.max_size	Maximum size of the octet sequences published by the OctetsDataWriters and received by the OctetsDataReader belonging to a DomainParticipant. Default: 2048
dds.builtin_type.keyed_octets.max_key_size	Maximum size of the keys used by the KeyedOctetsStringDataWriters and KeyedOctetsStringDataReaders belonging to a DomainParticipant (includes the NULL-terminated character). Default: 1024.
dds.builtin_type.keyed_octets.max_size	Maximum size of the octet sequences published by the KeyedOctetsDataWriters and received by the KeyedOctetsDataReaders belonging to a DomainParticipant. Default: 2048

For more information about the built-in types, including how to control memory usage and maximum lengths, please see the "Data Types and DDS Data Samples" chapter in the `User's Manual`.

7.13 Built-in Topic's Trust Types

Types used as part of **DDS_ParticipantBuiltinTopicData** (p. 1229), **DDS_PublicationBuiltinTopicData** (p. 1264), **DDS_SubscriptionBuiltinTopicData** (p. 1370) to describe Trust Plugins configuration.

Classes

- struct **DDS_ParticipantTrustProtectionInfo**

  Trust Plugins Protection information associated with the discovered DomainParticipant.
- struct **DDS_EndpointTrustProtectionInfo**

  Trust Plugins Protection information associated with the discovered endpoint.
- struct **DDS_TrustAlgorithmRequirements**

  Type to describe Trust Plugins algorithm requirements for an entity.
- struct **DDS_ParticipantTrustSignatureAlgorithmInfo**

  Trust Plugins signature algorithm information associated with the discovered DomainParticipant.
- struct **DDS_ParticipantTrustKeyEstablishmentAlgorithmInfo**

  Trust Plugins key establishment algorithm information associated with the discovered DomainParticipant.
- struct **DDS_ParticipantTrustInterceptorAlgorithmInfo**

  Trust Plugins interception algorithm information associated with the discovered DomainParticipant.
- struct **DDS_ParticipantTrustAlgorithmInfo**

  Trust Plugins algorithm information associated with the discovered DomainParticipant.
- struct **DDS_EndpointTrustInterceptorAlgorithmInfo**

  Trust Plugins interception algorithm information associated with the discovered endpoint.
- struct **DDS_EndpointTrustAlgorithmInfo**

  Trust Plugins algorithm information associated with the discovered endpoint.

Typedefs

- typedef **DDS_UnsignedLong** **DDS_ParticipantTrustAttributesMask**
- typedef **DDS_UnsignedLong** **DDS_PluginParticipantTrustAttributesMask**
- typedef struct **DDS_ParticipantTrustProtectionInfo** **DDS_ParticipantTrustProtectionInfo**

  Trust Plugins Protection information associated with the discovered DomainParticipant.
- typedef **DDS_UnsignedLong** **DDS_EndpointTrustAttributesMask**
- typedef **DDS_UnsignedLong** **DDS_PluginEndpointTrustAttributesMask**
- typedef **DDS_UnsignedLong** **DDS_VendorEndpointTrustAttributesMask**
- typedef struct **DDS_EndpointTrustProtectionInfo** **DDS_EndpointTrustProtectionInfo**

  Trust Plugins Protection information associated with the discovered endpoint.
- typedef **DDS_UnsignedLong** **DDS_TrustAlgorithmBit**
- typedef **DDS_UnsignedLong** **DDS_TrustAlgorithmSet**
- typedef struct **DDS_TrustAlgorithmRequirements** **DDS_TrustAlgorithmRequirements**

  Type to describe Trust Plugins algorithm requirements for an entity.
- typedef struct **DDS_ParticipantTrustSignatureAlgorithmInfo** **DDS_ParticipantTrustSignatureAlgorithmInfo**

  Trust Plugins signature algorithm information associated with the discovered DomainParticipant.
- typedef struct **DDS_ParticipantTrustKeyEstablishmentAlgorithmInfo** **DDS_ParticipantTrustKey↩ EstablishmentAlgorithmInfo**

  Trust Plugins key establishment algorithm information associated with the discovered DomainParticipant.

- typedef struct **DDS_ParticipantTrustInterceptorAlgorithmInfo** **DDS_ParticipantTrustInterceptor↩ AlgorithmInfo**

  Trust Plugins interception algorithm information associated with the discovered DomainParticipant.
- typedef struct **DDS_ParticipantTrustAlgorithmInfo** **DDS_ParticipantTrustAlgorithmInfo**

  Trust Plugins algorithm information associated with the discovered DomainParticipant.
- typedef struct **DDS_EndpointTrustInterceptorAlgorithmInfo** **DDS_EndpointTrustInterceptorAlgorithmInfo**

  Trust Plugins interception algorithm information associated with the discovered endpoint.
- typedef struct **DDS_EndpointTrustAlgorithmInfo** **DDS_EndpointTrustAlgorithmInfo**

  Trust Plugins algorithm information associated with the discovered endpoint.

7.13.1 Detailed Description

Types used as part of **DDS_ParticipantBuiltinTopicData** (p. 1229), **DDS_PublicationBuiltinTopicData** (p. 1264), **DDS_SubscriptionBuiltinTopicData** (p. 1370) to describe Trust Plugins configuration.

These types are used to describe how RTPS wire serialization, discovery, and liveliness interact with the loaded Trust Plugins.

The meaning of the contents of these types may vary depending on what Trust Plugins the DomainParticipant is using. For information about how these types interact with the RTI Security Plugins, please refer to the `RTI Security Plugins User's Manual`.

7.13.2 Typedef Documentation

7.13.2.1 DDS_ParticipantTrustAttributesMask

```
typedef DDS_UnsignedLong DDS_ParticipantTrustAttributesMask
```

7.13.2.2 DDS_PluginParticipantTrustAttributesMask

```
typedef DDS_UnsignedLong DDS_PluginParticipantTrustAttributesMask
```

7.13.2.3 DDS_ParticipantTrustProtectionInfo

```
typedef struct DDS_ParticipantTrustProtectionInfo DDS_ParticipantTrustProtectionInfo
```

Trust Plugins Protection information associated with the discovered DomainParticipant.

7.13.2.4 DDS_EndpointTrustAttributesMask

```
typedef DDS_UnsignedLong DDS_EndpointTrustAttributesMask
```

7.13.2.5 DDS_PluginEndpointTrustAttributesMask

```
typedef DDS_UnsignedLong DDS_PluginEndpointTrustAttributesMask
```

7.13.2.6 DDS_VendorEndpointTrustAttributesMask

```
typedef DDS_UnsignedLong DDS_VendorEndpointTrustAttributesMask
```

7.13.2.7 DDS_EndpointTrustProtectionInfo

```
typedef struct DDS_EndpointTrustProtectionInfo DDS_EndpointTrustProtectionInfo
```

Trust Plugins Protection information associated with the discovered endpoint.

7.13.2.8 DDS_TrustAlgorithmBit

```
typedef DDS_UnsignedLong DDS_TrustAlgorithmBit
```

7.13.2.9 DDS_TrustAlgorithmSet

```
typedef DDS_UnsignedLong DDS_TrustAlgorithmSet
```

7.13.2.10 DDS_TrustAlgorithmRequirements

```
typedef struct DDS_TrustAlgorithmRequirements DDS_TrustAlgorithmRequirements
```

Type to describe Trust Plugins algorithm requirements for an entity.

7.13.2.11 DDS_ParticipantTrustSignatureAlgorithmInfo

typedef struct **DDS_ParticipantTrustSignatureAlgorithmInfo** **DDS_ParticipantTrustSignatureAlgorithmInfo**

Trust Plugins signature algorithm information associated with the discovered DomainParticipant.

7.13.2.12 DDS_ParticipantTrustKeyEstablishmentAlgorithmInfo

typedef struct **DDS_ParticipantTrustKeyEstablishmentAlgorithmInfo** **DDS_ParticipantTrustKeyEstablishmentAlgorithmInfo**

Trust Plugins key establishment algorithm information associated with the discovered DomainParticipant.

7.13.2.13 DDS_ParticipantTrustInterceptorAlgorithmInfo

typedef struct **DDS_ParticipantTrustInterceptorAlgorithmInfo** **DDS_ParticipantTrustInterceptorAlgorithmInfo**

Trust Plugins interception algorithm information associated with the discovered DomainParticipant.

7.13.2.14 DDS_ParticipantTrustAlgorithmInfo

typedef struct **DDS_ParticipantTrustAlgorithmInfo** **DDS_ParticipantTrustAlgorithmInfo**

Trust Plugins algorithm information associated with the discovered DomainParticipant.

7.13.2.15 DDS_EndpointTrustInterceptorAlgorithmInfo

typedef struct **DDS_EndpointTrustInterceptorAlgorithmInfo** **DDS_EndpointTrustInterceptorAlgorithmInfo**

Trust Plugins interception algorithm information associated with the discovered endpoint.

7.13.2.16 DDS_EndpointTrustAlgorithmInfo

```
typedef struct DDS_EndpointTrustAlgorithmInfo DDS_EndpointTrustAlgorithmInfo
```

Trust Plugins algorithm information associated with the discovered endpoint.

7.14 Dynamic Data

<<***extension***>> *(*p. *432)* The Dynamic Data API provides a way to interact with arbitrarily complex data types at runtime without the need for code generation.

Classes

- struct **DDS_DynamicDataProperty_t**

  *A collection of attributes used to configure **DDS_DynamicData** (*p. *1013) objects.*
- struct **DDS_DynamicDataTypeSerializationProperty_t**

  Properties that govern how data of a certain type will be serialized on the network.
- struct **DDS_DynamicDataInfo**

  *A descriptor for a **DDS_DynamicData** (*p. *1013) object.*
- struct **DDS_DynamicDataMemberInfo**

  A descriptor for a single member (i.e. field) of dynamically defined data type.
- struct **DDS_DynamicData**

  A sample of any complex data type, which can be inspected and manipulated reflectively.
- struct **DDS_DynamicDataSeq**

  *An ordered collection of **DDS_DynamicData** (*p. *1013) elements.*
- struct **DDS_DynamicDataTypeProperty_t**

  *A collection of attributes used to configure **DDSDynamicDataTypeSupport** (*p. *1725) objects.*
- struct **DDS_DynamicDataJsonParserProperties_t**

  *A collection of attributes used to configure **DDS_DynamicData** (*p. *1013) objects.*
- class **DDSDynamicDataTypeSupport**

  *A factory for registering a dynamically defined type and creating **DDS_DynamicData** (*p. *1013) objects.*
- class **DDSDynamicDataReader**

  *Reads (subscribes to) objects of type **DDS_DynamicData** (*p. *1013).*
- class **DDSDynamicDataWriter**

  *Writes (publishes) objects of type **DDS_DynamicData** (*p. *1013).*

Macros

- #define **DDS_DYNAMIC_DATA_MEMBER_ID_UNSPECIFIED**

  A sentinel value that indicates that no member ID is needed in order to perform some operation.

Typedefs

- typedef **DDS_Long DDS_DynamicDataMemberId**

  An integer that uniquely identifies some member of a data type within that type.

Variables

- const struct **DDS_DynamicDataProperty_t DDS_DYNAMIC_DATA_PROPERTY_DEFAULT**

  *Sentinel constant indicating default values for **DDS_DynamicDataProperty_t** (p. 1137).*
- const struct **DDS_DynamicDataTypeProperty_t DDS_DYNAMIC_DATA_TYPE_PROPERTY_DEFAULT**

  *Sentinel constant indicating default values for **DDS_DynamicDataTypeProperty_t** (p. 1138).*
- struct **DDS_DynamicDataJsonParserProperties_t DDS_DYNAMIC_DATA_JSON_PARSER_PROPERTIES_DEFAULT**

  *Sentinel constant indicating default values for **DDS_DynamicDataProperty_t** (p. 1137).*

7.14.1 Detailed Description

<<***extension***>> *(p. 432)* The Dynamic Data API provides a way to interact with arbitrarily complex data types at runtime without the need for code generation.

This API allows you to define new data types, modify existing data types, and interact reflectively with samples. To use it, you will take the following steps:

1. Obtain a DDS_TypeCode (p. 1428) (see Type Code Support (p. 79)) that defines the type definition you want to use.
A **DDS_TypeCode** (p. 1428) includes a type's *kind* (**DDS_TCKind** (p. 89)), *name*, and *members* (that is, fields). You can create your own **DDS_TypeCode** (p. 1428) using the **DDS_TypeCodeFactory** (p. 1476) class – see, for example, the **DDS_TypeCodeFactory::create_struct_tc** (p. 1480) method. Alternatively, you can use a remote **DDS_TypeCode** (p. 1428) that you discovered on the network (see **Built-in Topics** (p. 61)) or one generated by rtiddsgen (see the `Code Generator User's Manual`). If you are relying on discovering TypeCodes on the network and are using TypeObject v2 (TypeObject v2 is the default behavior, see **DDS_DISCOVERYCONFIG_TYPE_LOOKUP_SERVICE_CHANNEL** (p. 592)), you must set **DDS_DiscoveryConfigQosPolicy::request_types_filter** (p. 959) to a string filter containing all of the topic names for which you want to dynamically discover types ("∗" to request all types). Otherwise, types will only be discovered if a local endpoint is already created on the topic and the local endpoint matches the remote endpoint.

2. Wrap the DDS_TypeCode (p. 1428) in a DDSDynamicDataTypeSupport (p. 1725) object.
See the constructor **DDSDynamicDataTypeSupport::DDSDynamicDataTypeSupport** (p. 1726). This object lets you connect the type definition to a **DDSDomainParticipant** (p. 1619) and manage data samples (of type **DDS_DynamicData** (p. 1013)).

3. Register the DDSDynamicDataTypeSupport (p. 1725) with one or more domain participants.
See **DDSDynamicDataTypeSupport::register_type** (p. 1727). This action associates the data type with a logical name that you can use to create topics. (Starting with this step, working with a dynamically defined data type is almost exactly the same as working with a generated one.)

4. Create a DDSTopic (p. 1885) from the DDSDomainParticipant (p. 1619).
Use the name under which you registered your data type – see **DDSDomainParticipant::create_topic** (p. 1650). This **DDSTopic** (p. 1885) is what you will use to produce and consume data.

5. Create a DDSDynamicDataWriter (p. 1730) and/or DDSDynamicDataReader (p. 1724).
These objects will produce and/or consume data (of type **DDS_DynamicData** (p. 1013)) on the **DDSTopic** (p. 1885). You can create these objects directly from the **DDSDomainParticipant** (p. 1619) – see **DDSDomainParticipant::create_datawriter** (p. 1683) and **DDSDomainParticipant::create_datareader** (p. 1687) – or by first creating intermediate **DDSPublisher** (p. 1818) and **DDSSubscriber** (p. 1860) objects – see **DDSDomainParticipant::create_publisher** (p. 1644) and **DDSDomainParticipant::create_subscriber** (p. 1647).

6. Write and/or read the data of interest.

7. Tear down the objects described above.
You should delete them in the reverse order in which you created them. Note that unregistering your data type with the **DDSDomainParticipant** (p. 1619) is optional; all types are automatically unregistered when the **DDSDomainParticipant** (p. 1619) itself is deleted.

7.14.2 Macro Definition Documentation

7.14.2.1 DDS_DYNAMIC_DATA_MEMBER_ID_UNSPECIFIED

```
#define DDS_DYNAMIC_DATA_MEMBER_ID_UNSPECIFIED
```

A sentinel value that indicates that no member ID is needed in order to perform some operation.

Most commonly, this constant will be used in "get" operations to indicate that a lookup should be performed based on a name, not on an ID.

See also

DDS_DynamicDataMemberId (p. 105)

7.14.3 Typedef Documentation

7.14.3.1 DDS_DynamicDataMemberId

```
typedef DDS_Long DDS_DynamicDataMemberId
```

An integer that uniquely identifies some member of a data type within that type.

The range of a member ID is the range of an unsigned short integer, except for the value 0, which is reserved.

See also

DDS_DYNAMIC_DATA_MEMBER_ID_UNSPECIFIED (p. 105)

7.14.4 Variable Documentation

7.14.4.1 DDS_DYNAMIC_DATA_PROPERTY_DEFAULT

```
const struct DDS_DynamicDataProperty_t DDS_DYNAMIC_DATA_PROPERTY_DEFAULT [extern]
```

Sentinel constant indicating default values for **DDS_DynamicDataProperty_t** (p. 1137).

Pass this object instead of your own **DDS_DynamicDataProperty_t** (p. 1137) object to use the default property values:

```
DDS_DynamicData* sample = new DDS_DynamicData(

    myTypeCode,

    &DDS_DYNAMIC_DATA_PROPERTY_DEFAULT);
```

See also

DDS_DynamicDataProperty_t (p. 1137)

Examples

HelloWorldPlugin.cxx.

7.14.4.2 DDS_DYNAMIC_DATA_TYPE_PROPERTY_DEFAULT

```
const struct DDS_DynamicDataTypeProperty_t DDS_DYNAMIC_DATA_TYPE_PROPERTY_DEFAULT [extern]
```

Sentinel constant indicating default values for **DDS_DynamicDataTypeProperty_t** (p. 1138).

Pass this object instead of your own **DDS_DynamicDataTypeProperty_t** (p. 1138) object to use the default property values:

```
DDS_DynamicDataTypeSupport* support = new DDS_DynamicDataTypeSupport(

    myTypeCode,

    &DDS_DYNAMIC_DATA_TYPE_PROPERTY_DEFAULT);
```

See also

DDS_DynamicDataTypeProperty_t (p. 1138)

7.14.4.3 DDS_DYNAMIC_DATA_JSON_PARSER_PROPERTIES_DEFAULT

```
struct DDS_DynamicDataJsonParserProperties_t DDS_DYNAMIC_DATA_JSON_PARSER_PROPERTIES_DEFAULT
[extern]
```

Sentinel constant indicating default values for **DDS_DynamicDataProperty_t** (p. 1137).

Pass this object instead of your own **DDS_DynamicDataProperty_t** (p. 1137) object to use the default property values:

```
DDS_DynamicData* sample = new DDS_DynamicData(

    myTypeCode,

    &DDS_DYNAMIC_DATA_PROPERTY_DEFAULT);
```

See also

DDS_DynamicDataProperty_t (p. 1137)

7.15 Publication Module

Contains the **DDSFlowController** (p. 1736), **DDSPublisher** (p. 1818), and **DDSDataWriter** (p. 1589) classes as well as the **DDSPublisherListener** (p. 1839) and **DDSDataWriterListener** (p. 1612) interfaces, and more generally, all that is needed on the publication side.

Modules

- **Publishers**

  DDSPublisher *(p. 1818) entity and associated elements*
- **Data Writers**

  DDSDataWriter *(p. 1589) entity and associated elements*
- **Flow Controllers**

  *<<**extension**>> (p. 432)* ***DDSFlowController*** *(p. 1736) and associated elements*
- **Multi-channel DataWriters**

  APIs related to Multi-channel DataWriters.

7.15.1 Detailed Description

Contains the **DDSFlowController** (p. 1736), **DDSPublisher** (p. 1818), and **DDSDataWriter** (p. 1589) classes as well as the **DDSPublisherListener** (p. 1839) and **DDSDataWriterListener** (p. 1612) interfaces, and more generally, all that is needed on the publication side.

"DCPS Publication package"

7.16 Publishers

DDSPublisher (p. 1818) entity and associated elements

Classes

- struct **DDS_PublisherQos**

  QoS policies supported by a ***DDSPublisher*** *(p. 1818) entity.*
- class **DDSPublisherListener**

  *<<**interface**>> (p. 433)* ***DDSListener*** *(p. 1794) for* ***DDSPublisher*** *(p. 1818) status.*
- class **DDSPublisherSeq**

  Declares IDL `sequence` *<* ***DDSPublisher*** *(p. 1818) > .*
- class **DDSPublisher**

  *<<**interface**>> (p. 433) A publisher is the object responsible for the actual dissemination of publications.*

Functions

- **DDS_Boolean DDS_PublisherQos_equals** (const struct **DDS_PublisherQos** *self, const struct **DDS_↩ PublisherQos** *other)

  *Compares two **DDS_PublisherQos** (p. 1276) for equality.*
- **DDS_ReturnCode_t DDS_PublisherQos::print** () const

  *Prints this **DDS_PublisherQos** (p. 1276) to stdout.*
- **DDS_ReturnCode_t DDS_PublisherQos::to_string** (char *string, **DDS_UnsignedLong** &string_size) const

  *Obtains a string representation of this **DDS_PublisherQos** (p. 1276).*
- **DDS_ReturnCode_t DDS_PublisherQos::to_string** (char *string, **DDS_UnsignedLong** &string_size, const **DDS_PublisherQos** &base) const

  *Obtains a string representation of this **DDS_PublisherQos** (p. 1276).*
- **DDS_ReturnCode_t DDS_PublisherQos::to_string** (char *string, **DDS_UnsignedLong** &string_size, const **DDS_QosPrintFormat** &format) const

  *Obtains a string representation of this **DDS_PublisherQos** (p. 1276).*
- **DDS_ReturnCode_t DDS_PublisherQos::to_string** (char *string, **DDS_UnsignedLong** &string_size, const **DDS_PublisherQos** &base, const **DDS_QosPrintFormat** &format) const

  *Obtains a string representation of this **DDS_PublisherQos** (p. 1276).*
- **DDS_ReturnCode_t DDS_PublisherQos::to_string** (char *string, **DDS_UnsignedLong** &string_size, const **DDS_QosPrintAll_t** &) const

  *Obtains a string representation of this **DDS_PublisherQos** (p. 1276).*
- **DDS_ReturnCode_t DDS_PublisherQos::to_string** (char *string, **DDS_UnsignedLong** &string_size, const **DDS_QosPrintAll_t** &, const **DDS_QosPrintFormat** &format) const

  *Obtains a string representation of this **DDS_PublisherQos** (p. 1276).*

Variables

- const struct **DDS_DataWriterQos DDS_DATAWRITER_QOS_DEFAULT**

  *Special value for creating **DDSDataWriter** (p. 1589) with default QoS.*
- const struct **DDS_DataWriterQos DDS_DATAWRITER_QOS_USE_TOPIC_QOS**

  *Special value for creating **DDSDataWriter** (p. 1589) with a combination of the default **DDS_DataWriterQos** (p. 909) and the **DDS_TopicQos** (p. 1397).*

7.16.1 Detailed Description

DDSPublisher (p. 1818) entity and associated elements

7.16.2 Function Documentation

7.16.2.1 DDS_PublisherQos_equals()

```
DDS_Boolean DDS_PublisherQos_equals (
            const struct  DDS_PublisherQos * self,
            const struct  DDS_PublisherQos * other )
```

Compares two **DDS_PublisherQos** (p. 1276) for equality.

Parameters

self	<<***in***>> *(*p. *433)* This PublisherQos.
other	<<***in***>> *(*p. *433)* The other PublisherQos to be compared with this PublisherQos.

Returns

DDS_BOOLEAN_TRUE (p. 513) if the two Qos objects are equal or **DDS_BOOLEAN_FALSE** (p. 513) otherwise.

Referenced by **DDS_PublisherQos::operator!=()**, and **DDS_PublisherQos::operator==()**.

7.16.2.2 print()

```
DDS_ReturnCode_t DDS_PublisherQos::print ( ) const  [inline]
```

Prints this **DDS_PublisherQos** (p. 1276) to stdout.

Only the differences between this **DDS_PublisherQos** (p. 1276) and the documented default are printed. If you wish to print everything regardless, see **DDS_PublisherQos::to_string(char∗, DDS_UnsignedLong&, const DDS_PublisherQos&, const DDS_QosPrintFormat&) const** (p. 111). The output is formatted according to the default values for **DDS_QosPrintFormat** (p. 1285).

Returns

DDS_RETCODE_OK (p. 532) if no error was encountered.

7.16.2.3 to_string() [1/6]

```
DDS_ReturnCode_t DDS_PublisherQos::to_string (
            char * string,
             DDS_UnsignedLong & string_size ) const  [inline]
```

Obtains a string representation of this **DDS_PublisherQos** (p. 1276).

Only the differences between this **DDS_PublisherQos** (p. 1276) and the documented default are printed to idref_PublisherQose string. If you wish to print everything regardless, see **DDS_PublisherQos::to_string(char∗, DDS_UnsignedLong&, const DDS_PublisherQos&, const DDS_QosPrintFormat&) const** (p. 111). The output is formatted according to the default values for **DDS_QosPrintFormat** (p. 1285).

If the supplied buffer is NULL, the required length of the string is returned via the string_size parameter.

If the supplied buffer is not NULL, the string representation of the **DDS_PublisherQos** (p. 1276) is written to the buffer.

Parameters

string	<<***out***>> *(p. 433)* The char buffer that will be used to store the string representation of this **DDS_PublisherQos** (p. 1276). If NULL, this function will return the required length of this buffer through the string_size parameter.
string_size	<<***inout***>> *(p. 433)* Cannot be NULL. The size of the supplied buffer. If the supplied buffer is NULL, or if the supplied buffer is not large enough, this value will be updated with the required length of the buffer.

Returns

DDS_RETCODE_OK (p. 532) if no error was encountered.

See also

DDS_PublisherQos::to_string(char*, DDS_UnsignedLong&, const DDS_PublisherQos&, const DDS_QosPrintFormat&) const (p. 111)

7.16.2.4 to_string() [2/6]

```
DDS_ReturnCode_t DDS_PublisherQos::to_string (
            char * string,
             DDS_UnsignedLong & string_size,
            const  DDS_PublisherQos & base ) const  [inline]
```

Obtains a string representation of this **DDS_PublisherQos** (p. 1276).

This overload behaves the same as **DDS_PublisherQos::to_string** (p. 109) but allows the caller to specify the **DDS_PublisherQos** (p. 1276), which is used as the base profile. Only the differences between the QoS and the base profile are included in the output string.

Parameters

string	<<***out***>> *(p. 433)* The char buffer that will be used to store the string representation of this **DDS_PublisherQos** (p. 1276). If NULL, this function will return the required length of this buffer through the string_size parameter.
string_size	<<***inout***>> *(p. 433)* Cannot be NULL. The size of the supplied buffer. If the supplied buffer is NULL, or if the supplied buffer is not large enough, this value will be updated with the required length of the buffer.
base	<<***in***>> *(p. 433)* The **DDS_PublisherQos** (p. 1276) to be used as the base QoS profile. Only the differences with respect to this base profile will be included in the output string.

Returns

DDS_RETCODE_OK (p. 532) if no error was encountered.

See also

DDS_PublisherQos::to_string(char*, DDS_UnsignedLong&, const DDS_PublisherQos&, const DDS_↩ QosPrintFormat&) const (p. 111)

References **DDS_QosPrintFormat_INITIALIZER**.

7.16.2.5 to_string() [3/6]

```
DDS_ReturnCode_t DDS_PublisherQos::to_string (
            char * string,
             DDS_UnsignedLong & string_size,
            const  DDS_QosPrintFormat & format ) const  [inline]
```

Obtains a string representation of this **DDS_PublisherQos** (p. 1276).

This overload behaves the same as **DDS_PublisherQos::to_string** (p. 109) but allows the caller to specify the **DDS↩ _QosPrintFormat** (p. 1285) which is used to format the output.

Parameters

string	<<***out***>> *(*p. *433)* The char buffer that will be used to store the string representation of this **DDS_PublisherQos** (p. 1276). If NULL, this function will return the required length of this buffer through the string_size parameter.
string_size	<<***inout***>> *(*p. *433)* Cannot be NULL. The size of the supplied buffer. If the supplied buffer is NULL, or if the supplied buffer is not large enough, this value will be updated with the required length of the buffer.
format	<<***in***>> *(*p. *433)* The **DDS_QosPrintFormat** (p. 1285) to be used to format the output string.

Returns

DDS_RETCODE_OK (p. 532) if no error was encountered.

See also

DDS_PublisherQos::to_string(char*, DDS_UnsignedLong&, const DDS_PublisherQos&, const DDS_↩ QosPrintFormat&) const (p. 111)

References **DDS_RETCODE_ERROR**, and **DDS_RETCODE_OK**.

7.16.2.6 to_string() [4/6]

```
DDS_ReturnCode_t DDS_PublisherQos::to_string (
            char * string,
             DDS_UnsignedLong & string_size,
            const  DDS_PublisherQos & base,
            const  DDS_QosPrintFormat & format ) const  [inline]
```

Obtains a string representation of this **DDS_PublisherQos** (p. 1276).

Only the differences between this **DDS_PublisherQos** (p. 1276) and the **DDS_PublisherQos** (p. 1276) supplied as the base are printed to the string.

If the supplied buffer is NULL, the required length of the string is returned via the string_size parameter.

If the supplied buffer is not NULL, the string representation of the **DDS_PublisherQos** (p. 1276) is written to the buffer.

Parameters

string	*<<**out**>> (p. 433)* The char buffer that will be used to store the string representation of this **DDS_PublisherQos** (p. 1276). If NULL, this function will return the required length of this buffer through the string_size parameter.
string_size	*<<**inout**>> (p. 433)* Cannot be NULL. The size of the supplied buffer. If the supplied buffer is NULL, or if the supplied buffer is not large enough, this value will be updated with the required length of the buffer.
base	*<<**in**>> (p. 433)* The **DDS_PublisherQos** (p. 1276) to be used as the base QoS profile. Only the differences with respect to this base profile will be included in the output string.
format	*<<**in**>> (p. 433)* The **DDS_QosPrintFormat** (p. 1285) to be used to format the output string.

Returns

DDS_RETCODE_OK (p. 532) if no error was encountered.

7.16.2.7 to_string() [5/6]

```
DDS_ReturnCode_t DDS_PublisherQos::to_string (
            char * string,
             DDS_UnsignedLong & string_size,
            const  DDS_QosPrintAll_t &  ) const  [inline]
```

Obtains a string representation of this **DDS_PublisherQos** (p. 1276).

This overload behaves the same as **DDS_PublisherQos::to_string** (p. 109) but prints the entire **DDS_PublisherQos** (p. 1276) object. The only valid value for the argument of type **DDS_QosPrintAll_t** (p. 1285) is **DDS_QOS_PRINT_ALL** (p. 169).

For example

```
PublisherQos qos;
char *str = NULL;
DDS_UnsignedLong str_size = 0;
qos.to_string(str, str_size, QOS_PRINT_ALL);
```

The resultant string is formatted according to the default value for **DDS_QosPrintFormat** (p. 1285).

Parameters

string	<<***out***>> *(*p. *433)* The char buffer that will be used to store the string representation of this **DDS_PublisherQos** (p. 1276). If NULL, this function will return the required length of this buffer through the string_size parameter.
string_size	<<***inout***>> *(*p. *433)* Cannot be NULL. The size of the supplied buffer. If the supplied buffer is NULL, or if the supplied buffer is not large enough, this value will be updated with the required length of the buffer.

Returns

DDS_RETCODE_OK (p. 532) if no error was encountered.

See also

DDS_PublisherQos::to_string(char∗, DDS_UnsignedLong&, const DDS_PublisherQos&, const DDS_↩ QosPrintFormat&) const (p. 111)

References **DDS_QosPrintFormat_INITIALIZER**.

7.16.2.8 to_string() [6/6]

```
DDS_ReturnCode_t DDS_PublisherQos::to_string (
            char * string,
             DDS_UnsignedLong & string_size,
            const  DDS_QosPrintAll_t & ,
            const  DDS_QosPrintFormat & format ) const  [inline]
```

Obtains a string representation of this **DDS_PublisherQos** (p. 1276).

This overload behaves the same as the **DDS_PublisherQos::to_string(char∗, DDS_UnsignedLong&, const DDS↩ _QosPrintAll_t&) const** (p. 112) overload. The only difference is that it allows the caller to specify the **DDS_Qos↩ PrintFormat** (p. 1285) which is used to format the resultant string.

Parameters

string	<<***out***>> *(*p. *433)* The char buffer that will be used to store the string representation of this **DDS_PublisherQos** (p. 1276). If NULL, this function will return the required length of this buffer through the string_size parameter.
string_size	<<***inout***>> *(*p. *433)* Cannot be NULL. The size of the supplied buffer. If the supplied buffer is NULL, or if the supplied buffer is not large enough, this value will be updated with the required length of the buffer.
format	<<***in***>> *(*p. *433)* The **DDS_QosPrintFormat** (p. 1285) to be used to format the output string.

Returns

DDS_RETCODE_OK (p. 532) if no error was encountered.

See also

DDS_PublisherQos::to_string(char*, DDS_UnsignedLong&, const DDS_PublisherQos&, const DDS_QosPrintFormat&) const (p. 111)

7.16.3 Variable Documentation

7.16.3.1 DDS_DATAWRITER_QOS_DEFAULT

```
const struct DDS_DataWriterQos DDS_DATAWRITER_QOS_DEFAULT [extern]
```

Special value for creating **DDSDataWriter** (p. 1589) with default QoS.

When used in **DDSPublisher::create_datawriter** (p. 1826), this special value is used to indicate that the **DDSDataWriter** (p. 1589) should be created with the default **DDSDataWriter** (p. 1589) QoS by means of the operation get_default_datawriter_qos and using the resulting QoS to create the **DDSDataWriter** (p. 1589).

When used in **DDSPublisher::set_default_datawriter_qos** (p. 1822), this special value is used to indicate that the default QoS should be reset back to the initial value that would be used if the **DDSPublisher::set_default_datawriter_qos** (p. 1822) operation had never been called.

When used in **DDSDataWriter::set_qos** (p. 1604), this special value is used to indicate that the QoS of the **DDSDataWriter** (p. 1589) should be changed to match the current defualt QoS set in the **DDSPublisher** (p. 1818) that the **DDSDataWriter** (p. 1589) belongs to.

Warning

This value is a constant and should never be modified. You cannot use this value to *get* the default QoS values for a DataWriter; for this purpose, use **DDSDomainParticipant::get_default_datawriter_qos** (p. 1626).

See also

DDSPublisher::create_datawriter (p. 1826)

DDSPublisher::set_default_datawriter_qos (p. 1822)

DDSDataWriter::set_qos (p. 1604)

7.16.3.2 DDS_DATAWRITER_QOS_USE_TOPIC_QOS

```
const struct DDS_DataWriterQos DDS_DATAWRITER_QOS_USE_TOPIC_QOS [extern]
```

Special value for creating **DDSDataWriter** (p. 1589) with a combination of the default **DDS_DataWriterQos** (p. 909) and the **DDS_TopicQos** (p. 1397).

The use of this value is equivalent to the application obtaining the default **DDS_DataWriterQos** (p. 909) and the **DDS_TopicQos** (p. 1397) (by means of the operation **DDSTopic::get_qos** (p. 1889)) and then combining these two QoS using the operation **DDSPublisher::copy_from_topic_qos** (p. 1834) whereby any policy that is set on the **DDS_TopicQos** (p. 1397) "overrides" the corresponding policy on the default QoS. The resulting QoS is then applied to the creation of the **DDSDataWriter** (p. 1589).

This value should only be used in **DDSPublisher::create_datawriter** (p. 1826).

See also

DDSPublisher::create_datawriter (p. 1826)

DDSPublisher::get_default_datawriter_qos (p. 1821)

DDSTopic::get_qos (p. 1889)

DDSPublisher::copy_from_topic_qos (p. 1834)

7.17 Data Writers

DDSDataWriter (p. 1589) entity and associated elements

Classes

- struct **DDS_OfferedDeadlineMissedStatus**

  DDS_OFFERED_DEADLINE_MISSED_STATUS *(p. 539)*
- struct **DDS_LivelinessLostStatus**

  DDS_LIVELINESS_LOST_STATUS *(p. 542)*
- struct **DDS_OfferedIncompatibleQosStatus**

  DDS_OFFERED_INCOMPATIBLE_QOS_STATUS *(p. 540)*
- struct **DDS_PublicationMatchedStatus**

  DDS_PUBLICATION_MATCHED_STATUS *(p. 543)*
- struct **DDS_ServiceRequestAcceptedStatus**

  DDS_SERVICE_REQUEST_ACCEPTED_STATUS *(p. 544)*
- struct **DDS_ReliableWriterCacheEventCount**

  *<<**extension**>> (p. 432) The number of times the number of unacknowledged samples in the cache of a reliable writer hit a certain well-defined threshold.*
- struct **DDS_ReliableWriterCacheChangedStatus**

  *<<**extension**>> (p. 432) A summary of the state of a data writer's cache of unacknowledged samples written.*
- struct **DDS_ReliableReaderActivityChangedStatus**

  *<<**extension**>> (p. 432) Describes the activity (i.e. are acknowledgements forthcoming) of reliable readers matched to a reliable writer.*

- struct **DDS_DataWriterCacheStatus**

  *<<**extension**>> (p. 432) The status of the DataWriter's cache. Provides information on cache related metrics such as the number of samples and instances in the DataWriter queue.*
- struct **DDS_DataWriterProtocolStatus**

  *<<**extension**>> (p. 432) The status of a writer's internal protocol related metrics, like the number of samples pushed, pulled, filtered; and status of wire protocol traffic.*
- struct **DDS_AcknowledgmentInfo**

  Information about an application-level acknowledged sample.
- struct **DDS_DataWriterQos**

  *QoS policies supported by a **DDSDataWriter** (p. 1589) entity.*
- class **FooDataWriter**

  *<<**interface**>> (p. 433) <<**generic**>> (p. 433) User data type specific data writer.*
- class **DDSDataWriterListener**

  *<<**interface**>> (p. 433) **DDSListener** (p. 1794) for writer status.*
- class **DDSDataWriter**

  *<<**interface**>> (p. 433) Allows an application to set the value of the data to be published under a given **DDSTopic** (p. 1885).*

Functions

- **DDS_Boolean** **DDS_DataWriterQos_equals** (const struct **DDS_DataWriterQos** *self, const struct **DDS_DataWriterQos** *other)

  *Compares two **DDS_DataWriterQos** (p. 909) for equality.*
- **DDS_ReturnCode_t** **DDS_DataWriterQos::print** () const

  *Prints this **DDS_DataWriterQos** (p. 909) to stdout.*
- **DDS_ReturnCode_t** **DDS_DataWriterQos::to_string** (char *string, **DDS_UnsignedLong** &string_size) const

  *Obtains a string representation of this **DDS_DataWriterQos** (p. 909).*
- **DDS_ReturnCode_t** **DDS_DataWriterQos::to_string** (char *string, **DDS_UnsignedLong** &string_size, const **DDS_DataWriterQos** &base) const

  *Obtains a string representation of this **DDS_DataWriterQos** (p. 909).*
- **DDS_ReturnCode_t** **DDS_DataWriterQos::to_string** (char *string, **DDS_UnsignedLong** &string_size, const **DDS_QosPrintFormat** &format) const

  *Obtains a string representation of this **DDS_DataWriterQos** (p. 909).*
- **DDS_ReturnCode_t** **DDS_DataWriterQos::to_string** (char *string, **DDS_UnsignedLong** &string_size, const **DDS_DataWriterQos** &base, const **DDS_QosPrintFormat** &format) const

  *Obtains a string representation of this **DDS_DataWriterQos** (p. 909).*
- **DDS_ReturnCode_t** **DDS_DataWriterQos::to_string** (char *string, **DDS_UnsignedLong** &string_size, const **DDS_QosPrintAll_t** &) const

  *Obtains a string representation of this **DDS_DataWriterQos** (p. 909).*
- **DDS_ReturnCode_t** **DDS_DataWriterQos::to_string** (char *string, **DDS_UnsignedLong** &string_size, const **DDS_QosPrintAll_t** &, const **DDS_QosPrintFormat** &format) const

  *Obtains a string representation of this **DDS_DataWriterQos** (p. 909).*

7.17.1 Detailed Description

DDSDataWriter (p. 1589) entity and associated elements

7.17.2 Function Documentation

7.17.2.1 DDS_DataWriterQos_equals()

```
DDS_Boolean DDS_DataWriterQos_equals (
            const struct  DDS_DataWriterQos * self,
            const struct  DDS_DataWriterQos * other )
```

Compares two **DDS_DataWriterQos** (p. 909) for equality.

Parameters

self	<<***in***>> *(p. 433)* This DataWriterQos.
other	<<***in***>> *(p. 433)* The other DataWriterQos to be compared with this DataWriterQos.

Returns

DDS_BOOLEAN_TRUE (p. 513) if the two Qos objects are equal or **DDS_BOOLEAN_FALSE** (p. 513) otherwise.

Referenced by **DDS_DataWriterQos::operator!=()**, and **DDS_DataWriterQos::operator==()**.

7.17.2.2 print()

```
DDS_ReturnCode_t DDS_DataWriterQos::print ( ) const  [inline]
```

Prints this **DDS_DataWriterQos** (p. 909) to stdout.

Only the differences between this **DDS_DataWriterQos** (p. 909) and the documented default are printed. If you wish to print everything regardless, see **DDS_DataWriterQos::to_string(char*, DDS_UnsignedLong&, const DDS_DataWriterQos&, const DDS_QosPrintFormat&) const** (p. 120). The output is formatted according to the default values for **DDS_QosPrintFormat** (p. 1285).

Returns

DDS_RETCODE_OK (p. 532) if no error was encountered.

7.17.2.3 to_string() [1/6]

```
DDS_ReturnCode_t DDS_DataWriterQos::to_string (
            char * string,
             DDS_UnsignedLong & string_size ) const  [inline]
```

Obtains a string representation of this **DDS_DataWriterQos** (p. 909).

Only the differences between this **DDS_DataWriterQos** (p. 909) and the documented default are printed to the string. If you wish to print everything regardless, see **DDS_DataWriterQos::to_string(char∗, DDS_UnsignedLong&, const DDS_DataWriterQos&, const DDS_QosPrintFormat&) const** (p. 120). The output is formatted according to the default values for **DDS_QosPrintFormat** (p. 1285).

If the supplied buffer is NULL, the required length of the string is returned via the string_size parameter.

If the supplied buffer is not NULL, the string representation of the **DDS_DataWriterQos** (p. 909) is written to the buffer.

Parameters

string	*<<**out**>> (p. 433)* The char buffer that will be used to store the string representation of this **DDS_DataWriterQos** (p. 909). If NULL, this function will return the required length of this buffer through the string_size parameter.
string_size	*<<**inout**>> (p. 433)* Cannot be NULL. The size of the supplied buffer. If the supplied buffer is NULL, or if the supplied buffer is not large enough, this value will be updated with the required length of the buffer.

Returns

DDS_RETCODE_OK (p. 532) if no error was encountered.

See also

DDS_DataWriterQos::to_string(char∗, DDS_UnsignedLong&, const DDS_DataWriterQos&, const DDS_↩ QosPrintFormat&) const (p. 120)

7.17.2.4 to_string() [2/6]

```
DDS_ReturnCode_t DDS_DataWriterQos::to_string (
            char * string,
             DDS_UnsignedLong & string_size,
            const  DDS_DataWriterQos & base ) const  [inline]
```

Obtains a string representation of this **DDS_DataWriterQos** (p. 909).

This overload behaves the same as **DDS_DataWriterQos::to_string** (p. 117) but allows the caller to specify the **DDS↩ _DataWriterQos** (p. 909), which is used as the base profile. Only the differences between the QoS and the base profile are included in the output string.

Parameters

string	<<***out***>> *(*p. *433)* The char buffer that will be used to store the string representation of this **DDS_DataWriterQos** (p. 909). If NULL, this function will return the required length of this buffer through the string_size parameter.
string_size	<<***inout***>> *(*p. *433)* Cannot be NULL. The size of the supplied buffer. If the supplied buffer is NULL, or if the supplied buffer is not large enough, this value will be updated with the required length of the buffer.
base	<<***in***>> *(*p. *433)* The **DDS_DataWriterQos** (p. 909) to be used as the base QoS profile. Only the differences with respect to this base profile will be included in the output string.

Returns

DDS_RETCODE_OK (p. 532) if no error was encountered.

See also

DDS_DataWriterQos::to_string(char*, DDS_UnsignedLong&, const DDS_DataWriterQos&, const DDS_↩ QosPrintFormat&) const (p. 120)

References **DDS_QosPrintFormat_INITIALIZER**.

7.17.2.5 to_string() [3/6]

```
DDS_ReturnCode_t DDS_DataWriterQos::to_string (
            char * string,
             DDS_UnsignedLong & string_size,
            const  DDS_QosPrintFormat & format ) const  [inline]
```

Obtains a string representation of this **DDS_DataWriterQos** (p. 909).

This overload behaves the same as **DDS_DataWriterQos::to_string** (p. 117) but allows the caller to specify the **DDS↩ _QosPrintFormat** (p. 1285) which is used to format the output.

Parameters

string	<<***out***>> *(*p. *433)* The char buffer that will be used to store the string representation of this **DDS_DataWriterQos** (p. 909). If NULL, this function will return the required length of this buffer through the string_size parameter.
string_size	<<***inout***>> *(*p. *433)* Cannot be NULL. The size of the supplied buffer. If the supplied buffer is NULL, or if the supplied buffer is not large enough, this value will be updated with the required length of the buffer.
format	<<***in***>> *(*p. *433)* The **DDS_QosPrintFormat** (p. 1285) to be used to format the output string.

Returns

DDS_RETCODE_OK (p. 532) if no error was encountered.

See also

DDS_DataWriterQos::to_string(char*, DDS_UnsignedLong&, const DDS_DataWriterQos&, const DDS_↩ QosPrintFormat&) const (p. 120)

References **DDS_RETCODE_ERROR**, and **DDS_RETCODE_OK**.

7.17.2.6 to_string() [4/6]

```
DDS_ReturnCode_t DDS_DataWriterQos::to_string (
            char * string,
             DDS_UnsignedLong & string_size,
            const  DDS_DataWriterQos & base,
            const  DDS_QosPrintFormat & format ) const  [inline]
```

Obtains a string representation of this **DDS_DataWriterQos** (p. 909).

Only the differences between this **DDS_DataWriterQos** (p. 909) and the **DDS_DataWriterQos** (p. 909) supplied as the base are printed to the string.

If the supplied buffer is NULL, the required length of the string is returned via the string_size parameter.

If the supplied buffer is not NULL, the string representation of the **DDS_DataWriterQos** (p. 909) is written to the buffer.

Parameters

string	*<<**out**>> (p. 433)* The char buffer that will be used to store the string representation of this **DDS_DataWriterQos** (p. 909). If NULL, this function will return the required length of this buffer through the string_size parameter.
string_size	*<<**inout**>> (p. 433)* Cannot be NULL. The size of the supplied buffer. If the supplied buffer is NULL, or if the supplied buffer is not large enough, this value will be updated with the required length of the buffer.
base	*<<**in**>> (p. 433)* The **DDS_DataWriterQos** (p. 909) to be used as the base QoS profile. Only the differences with respect to this base profile will be included in the output string.
format	*<<**in**>> (p. 433)* The **DDS_QosPrintFormat** (p. 1285) to be used to format the output string.

Returns

DDS_RETCODE_OK (p. 532) if no error was encountered.

7.17.2.7 to_string() [5/6]

```
DDS_ReturnCode_t DDS_DataWriterQos::to_string (
            char * string,
             DDS_UnsignedLong & string_size,
            const  DDS_QosPrintAll_t &  ) const  [inline]
```

Obtains a string representation of this **DDS_DataWriterQos** (p. 909).

This overload behaves the same as **DDS_DataWriterQos::to_string** (p. 117) but prints the entire **DDS_DataWriterQos** (p. 909) object. The only valid value for the argument of type **DDS_QosPrintAll_t** (p. 1285) is **DDS_QOS_PRINT_ALL** (p. 169).

For example
```
DataWriterQos qos;
char *str = NULL;
DDS_UnsignedLong str_size = 0;
qos.to_string(str, str_size, QOS_PRINT_ALL);
```

The resultant string is formatted according to the default value for **DDS_QosPrintFormat** (p. 1285).

Parameters

string	<<***out***>> *(p. 433)* The char buffer that will be used to store the string representation of this **DDS_DataWriterQos** (p. 909). If NULL, this function will return the required length of this buffer through the string_size parameter.
string_size	<<***inout***>> *(p. 433)* Cannot be NULL. The size of the supplied buffer. If the supplied buffer is NULL, or if the supplied buffer is not large enough, this value will be updated with the required length of the buffer.

Returns

DDS_RETCODE_OK (p. 532) if no error was encountered.

See also

DDS_DataWriterQos::to_string(char∗, DDS_UnsignedLong&, const DDS_DataWriterQos&, const DDS_↩ QosPrintFormat&) const (p. 120)

References **DDS_QosPrintFormat_INITIALIZER**.

7.17.2.8 to_string() [6/6]

```
DDS_ReturnCode_t DDS_DataWriterQos::to_string (
            char * string,
             DDS_UnsignedLong & string_size,
            const  DDS_QosPrintAll_t & ,
            const  DDS_QosPrintFormat & format ) const  [inline]
```

Obtains a string representation of this **DDS_DataWriterQos** (p. 909).

This overload behaves the same as the **DDS_DataWriterQos::to_string(char∗, DDS_UnsignedLong&, const DDS↩ _QosPrintAll_t&) const** (p. 120) overload. The only difference is that it allows the caller to specify the **DDS_Qos↩ PrintFormat** (p. 1285) which is used to format the resultant string.

Parameters

string	<<***out***>> *(*p. *433)* The char buffer that will be used to store the string representation of this **DDS_DataWriterQos** (p. 909). If NULL, this function will return the required length of this buffer through the string_size parameter.
string_size	<<***inout***>> *(*p. *433)* Cannot be NULL. The size of the supplied buffer. If the supplied buffer is NULL, or if the supplied buffer is not large enough, this value will be updated with the required length of the buffer.
format	<<***in***>> *(*p. *433)* The **DDS_QosPrintFormat** (p. 1285) to be used to format the output string.

Returns

DDS_RETCODE_OK (p. 532) if no error was encountered.

See also

DDS_DataWriterQos::to_string(char∗, DDS_UnsignedLong&, const DDS_DataWriterQos&, const DDS_↩ QosPrintFormat&) const (p. 120)

7.18 Flow Controllers

<<***extension***>> *(*p. *432)* **DDSFlowController** (p. 1736) and associated elements

Classes

- struct **DDS_FlowControllerTokenBucketProperty_t**

  DDSFlowController *(*p. *1736) uses the popular token bucket approach for open loop network flow control. The flow control characteristics are determined by the token bucket properties.*
- struct **DDS_FlowControllerProperty_t**

  Determines the flow control characteristics of the ***DDSFlowController*** *(*p. *1736).*
- class **DDSFlowController**

  <<***interface***>> *(*p. *433) A flow controller is the object responsible for shaping the network traffic by determining when attached asynchronous* ***DDSDataWriter*** *(*p. *1589) instances are allowed to write data.*

Enumerations

- enum **DDS_FlowControllerSchedulingPolicy** {
  DDS_RR_FLOW_CONTROLLER_SCHED_POLICY ,
  DDS_EDF_FLOW_CONTROLLER_SCHED_POLICY ,
  DDS_HPF_FLOW_CONTROLLER_SCHED_POLICY }

  Kinds of flow controller scheduling policy.

Variables

- char * **DDS_DEFAULT_FLOW_CONTROLLER_NAME**

  [default] *Special value of* ***DDS_PublishModeQosPolicy::flow_controller_name*** *(p. 1281) that refers to the built-in default flow controller.*
- char * **DDS_FIXED_RATE_FLOW_CONTROLLER_NAME**

  Special value of ***DDS_PublishModeQosPolicy::flow_controller_name*** *(p. 1281) that refers to the built-in fixed-rate flow controller.*
- char * **DDS_ON_DEMAND_FLOW_CONTROLLER_NAME**

  Special value of ***DDS_PublishModeQosPolicy::flow_controller_name*** *(p. 1281) that refers to the built-in on-demand flow controller.*

7.18.1 Detailed Description

<<***extension***>> *(p. 432)* **DDSFlowController** (p. 1736) and associated elements

DDSFlowController (p. 1736) provides the network traffic shaping capability to asynchronous **DDSDataWriter** (p. 1589) instances. For use cases and advantages of publishing asnychronously, please refer to **DDS_PublishModeQosPolicy** (p. 1279) of **DDS_DataWriterQos** (p. 909).

See also

DDS_PublishModeQosPolicy (p. 1279)

DDS_DataWriterQos::publish_mode (p. 915)

DDS_AsynchronousPublisherQosPolicy (p. 798)

7.18.2 Enumeration Type Documentation

7.18.2.1 DDS_FlowControllerSchedulingPolicy

enum **DDS_FlowControllerSchedulingPolicy**

Kinds of flow controller scheduling policy.

Samples written by an asynchronous **DDSDataWriter** (p. 1589) are not sent in the context of the **FooDataWriter::write** (p. 1951) call. Instead, the middleware puts the samples in a queue for future processing. The **DDSFlowController** (p. 1736) associated with each asynchronous DataWriter instance determines when the samples are actually sent.

Each **DDSFlowController** (p. 1736) maintains a separate FIFO queue for each unique destination (remote application). Samples written by asynchronous **DDSDataWriter** (p. 1589) instances associated with the flow controller, are placed in the queues that correspond to the intended destinations of the sample.

When tokens become available, a flow controller must decide which queue(s) to grant tokens first. This is determined by the flow controller's scheduling policy. Once a queue has been granted tokens, it is serviced by the asynchronous publishing thread. The queued up samples will be coalesced and sent to the corresponding destination. The number of samples sent depends on the data size and the number of tokens granted.

QoS:

DDS_FlowControllerProperty_t (p. 1156)

Enumerator

DDS_RR_FLOW_CONTROLLER_SCHED_POLICY	Indicates to flow control in a round-robin fashion. Whenever tokens become available, the flow controller distributes the tokens uniformly across all of its (non-empty) destination queues. No destinations are prioritized. Instead, all destinations are treated equally and are serviced in a round-robin fashion.
DDS_EDF_FLOW_CONTROLLER_SCHED_POLICY	Indicates to flow control in an earliest-deadline-first fashion. A sample's deadline is determined by the time it was written plus the latency budget of the DataWriter at the time of the write call (as specified in the **DDS_LatencyBudgetQosPolicy** (p. 1173)). The relative priority of a flow controller's destination queue is determined by the earliest deadline across all samples it contains. When tokens become available, the **DDSFlowController** (p. 1736) distributes tokens to the destination queues in order of their deadline priority. In other words, the queue containing the sample with the earliest deadline is serviced first. The number of tokens granted equals the number of tokens required to send the first sample in the queue. Note that the priority of a queue may change as samples are sent (i.e. removed from the queue). If a sample must be sent to multiple destinations or two samples have an equal deadline value, the corresponding destination queues are serviced in a round-robin fashion. Hence, under the default **DDS_LatencyBudgetQosPolicy::duration** (p. 1175) setting, an `EDF_FLOW_CONTROLLER_SCHED_POLICY` **DDSFlowController** (p. 1736) preserves the order in which the user calls **FooDataWriter::write** (p. 1951) across the DataWriters associated with the flow controller. Since the **DDS_LatencyBudgetQosPolicy** (p. 1173) is mutable, a sample written second may contain an earlier deadline than the sample written first if the **DDS_LatencyBudgetQosPolicy::duration** (p. 1175) value is sufficiently decreased in between writing the two samples. In that case, if the first sample is not yet written (still in queue waiting for its turn), it inherits the priority corresponding to the (earlier) deadline from the second sample. In other words, the priority of a destination queue is always determined by the earliest deadline among all samples contained in the queue. This priority inheritance approach is required in order to both honor the updated **DDS_LatencyBudgetQosPolicy::duration** (p. 1175) and adhere to the **DDSDataWriter** (p. 1589) in-order data delivery guarantee. **[default]** for **DDSDataWriter** (p. 1589)

Enumerator

DDS_HPF_FLOW_CONTROLLER_SCHED_POLICY	Indicates to flow control in a highest-priority-first fashion. Determines the next destination queue to service as determined by the publication priority of the **DDSDataWriter** (p. 1589), channel of multi-channel DataWriter, or individual sample. The relative priority of a flow controller's destination queue is determined by the highest publication priority of all samples it contains. When tokens become available, the **DDSFlowController** (p. 1736) distributes tokens to the destination queues in order of their publication priority. In other words, the queue containing the sample with the highest publication priority is serviced first. The number of tokens granted equals the number of tokens required to send the first sample in the queue. Note that the priority of a queue may change as samples are sent (i.e. removed from the queue). If a sample must be sent to multiple destinations or two samples have an equal publication priority, the corresponding destination queues are serviced in a round-robin fashion. This priority inheritance approach is required in order to both honor the designated publication priority and adhere to the **DDSDataWriter** (p. 1589) in-order data delivery guarantee.

7.18.3 Variable Documentation

7.18.3.1 DDS_DEFAULT_FLOW_CONTROLLER_NAME

```
char* DDS_DEFAULT_FLOW_CONTROLLER_NAME  [extern]
```

[default] Special value of **DDS_PublishModeQosPolicy::flow_controller_name** (p. 1281) that refers to the built-in default flow controller.

RTI Connext provides several built-in **DDSFlowController** (p. 1736) for use with an asynchronous **DDSDataWriter** (p. 1589). The user can choose to use the built-in flow controllers and optionally modify their properties or can create a custom flow controller.

By default, flow control is disabled. That is, the built-in **DDS_DEFAULT_FLOW_CONTROLLER_NAME** (p. 125) flow controller does not apply any flow control. Instead, it allows data to be sent asynchronously as soon as it is written by the **DDSDataWriter** (p. 1589).

Essentially, this is equivalent to a user-created **DDSFlowController** (p. 1736) with the following **DDS_FlowController↩ Property_t** (p. 1156) settings:

- **DDS_FlowControllerProperty_t::scheduling_policy** (p. 1157) = **DDS_EDF_FLOW_CONTROLLER_SCHED_POLICY** (p. 124)
- **DDS_FlowControllerProperty_t::token_bucket** (p. 1157) `max_tokens` = **DDS_LENGTH_UNLIMITED** (p. 636)
- **DDS_FlowControllerProperty_t::token_bucket** (p. 1157) `tokens_added_per_period` = **DDS_LENGTH_UNLIMITED** (p. 636)
- **DDS_FlowControllerProperty_t::token_bucket** (p. 1157) `tokens_leaked_per_period` = 0
- **DDS_FlowControllerProperty_t::token_bucket** (p. 1157) `period` = 60 seconds
- **DDS_FlowControllerProperty_t::token_bucket** (p. 1157) `bytes_per_token` = **DDS_LENGTH_UNLIMITED** (p. 636)

See also

DDSPublisher::create_datawriter (p. 1826)

DDSDomainParticipant::lookup_flowcontroller (p. 1660)

DDSFlowController::set_property (p. 1737)

DDS_PublishModeQosPolicy (p. 1279)

DDS_AsynchronousPublisherQosPolicy (p. 798)

7.18.3.2 DDS_FIXED_RATE_FLOW_CONTROLLER_NAME

```
char* DDS_FIXED_RATE_FLOW_CONTROLLER_NAME  [extern]
```

Special value of **DDS_PublishModeQosPolicy::flow_controller_name** (p. 1281) that refers to the built-in fixed-rate flow controller.

RTI Connext provides several builtin **DDSFlowController** (p. 1736) for use with an asynchronous **DDSDataWriter** (p. 1589). The user can choose to use the built-in flow controllers and optionally modify their properties or can create a custom flow controller.

The built-in **DDS_FIXED_RATE_FLOW_CONTROLLER_NAME** (p. 126) flow controller shapes the network traffic by allowing data to be sent only once every second. Any accumulated samples destined for the same destination are coalesced into as few network packets as possible.

Essentially, this is equivalent to a user-created **DDSFlowController** (p. 1736) with the following **DDS_FlowControllerProperty_t** (p. 1156) settings:

- **DDS_FlowControllerProperty_t::scheduling_policy** (p. 1157) = **DDS_EDF_FLOW_CONTROLLER_SCHED_POLICY** (p. 124)
- **DDS_FlowControllerProperty_t::token_bucket** (p. 1157) `max_tokens` = **DDS_LENGTH_UNLIMITED** (p. 636)
- **DDS_FlowControllerProperty_t::token_bucket** (p. 1157) `tokens_added_per_period` = **DDS_LENGTH_UNLIMITED** (p. 636)
- **DDS_FlowControllerProperty_t::token_bucket** (p. 1157) `tokens_leaked_per_period` = **DDS_LENGTH_UNLIMITED** (p. 636)
- **DDS_FlowControllerProperty_t::token_bucket** (p. 1157) `period` = 1 second
- **DDS_FlowControllerProperty_t::token_bucket** (p. 1157) `bytes_per_token` = **DDS_LENGTH_UNLIMITED** (p. 636)

See also

DDSPublisher::create_datawriter (p. 1826)

DDSDomainParticipant::lookup_flowcontroller (p. 1660)

DDSFlowController::set_property (p. 1737)

DDS_PublishModeQosPolicy (p. 1279)

DDS_AsynchronousPublisherQosPolicy (p. 798)

7.18.3.3 DDS_ON_DEMAND_FLOW_CONTROLLER_NAME

```
char* DDS_ON_DEMAND_FLOW_CONTROLLER_NAME  [extern]
```

Special value of **DDS_PublishModeQosPolicy::flow_controller_name** (p. 1281) that refers to the built-in on-demand flow controller.

RTI Connext provides several builtin **DDSFlowController** (p. 1736) for use with an asynchronous **DDSDataWriter** (p. 1589). The user can choose to use the built-in flow controllers and optionally modify their properties or can create a custom flow controller.

The built-in **DDS_ON_DEMAND_FLOW_CONTROLLER_NAME** (p. 127) allows data to be sent only when the user calls **DDSFlowController::trigger_flow** (p. 1738). With each trigger, all accumulated data since the previous trigger is sent (across all **DDSPublisher** (p. 1818) or **DDSDataWriter** (p. 1589) instances). In other words, the network traffic shape is fully controlled by the user. Any accumulated samples destined for the same destination are coalesced into as few network packets as possible.

This external trigger source is ideal for users who want to implement some form of closed-loop flow control or who want to only put data on the wire every so many samples (e.g. with the number of samples based on **NDDS_Transport_Property_t::gather_send_buffer_count_max** (p. 2047)).

Essentially, this is equivalent to a user-created **DDSFlowController** (p. 1736) with the following **DDS_FlowControllerProperty_t** (p. 1156) settings:

- **DDS_FlowControllerProperty_t::scheduling_policy** (p. 1157) = **DDS_EDF_FLOW_CONTROLLER_SCHED_POLICY** (p. 124)
- **DDS_FlowControllerProperty_t::token_bucket** (p. 1157) `max_tokens` = **DDS_LENGTH_UNLIMITED** (p. 636)
- **DDS_FlowControllerProperty_t::token_bucket** (p. 1157) `tokens_added_per_period` = **DDS_LENGTH_UNLIMITED** (p. 636)
- **DDS_FlowControllerProperty_t::token_bucket** (p. 1157) `tokens_leaked_per_period` = **DDS_LENGTH_UNLIMITED** (p. 636)
- **DDS_FlowControllerProperty_t::token_bucket** (p. 1157) `period` = **DDS_DURATION_INFINITE** (p. 522)
- **DDS_FlowControllerProperty_t::token_bucket** (p. 1157) `bytes_per_token` = **DDS_LENGTH_UNLIMITED** (p. 636)

See also

DDSPublisher::create_datawriter (p. 1826)

DDSDomainParticipant::lookup_flowcontroller (p. 1660)

DDSFlowController::trigger_flow (p. 1738)

DDSFlowController::set_property (p. 1737)

DDS_PublishModeQosPolicy (p. 1279)

DDS_AsynchronousPublisherQosPolicy (p. 798)

7.19 Subscription Module

Contains the **DDSSubscriber** (p. 1860), **DDSDataReader** (p. 1556), **DDSReadCondition** (p. 1842), **DDSQueryCondition** (p. 1841), and **DDSTopicQuery** (p. 1895) classes, as well as the **DDSSubscriberListener** (p. 1881) and **DDSDataReaderListener** (p. 1583) interfaces, and more generally, all that is needed on the subscription side.

Modules

- **Subscribers**

  DDSSubscriber *(p. 1860) entity and associated elements*
- **DataReaders**

  DDSDataReader *(p. 1556) entity and associated elements*
- **Data Samples**

  DDS_SampleInfo *(p. 1344),* ***DDS_SampleStateKind*** *(p. 161),* ***DDS_ViewStateKind*** *(p. 163),* ***DDS_InstanceStateKind*** *(p. 165) and associated elements*
- **SampleProcessor**

  *<<**experimental**>> (p. 432) <<**extension**>> (p. 432) Utility to concurrently read and process the data samples received by* ***DDSDataReader*** *(p. 1556).*

7.19.1 Detailed Description

Contains the **DDSSubscriber** (p. 1860), **DDSDataReader** (p. 1556), **DDSReadCondition** (p. 1842), **DDSQuery↩ Condition** (p. 1841), and **DDSTopicQuery** (p. 1895) classes, as well as the **DDSSubscriberListener** (p. 1881) and **DDSDataReaderListener** (p. 1583) interfaces, and more generally, all that is needed on the subscription side.

"DCPS Subscription package"

7.19.2 Access to data samples

Data is made available to the application by the following operations on **DDSDataReader** (p. 1556) objects: **FooData↩ Reader::read** (p. 1919), **FooDataReader::read_w_condition** (p. 1925), **FooDataReader::take** (p. 1920), **FooData↩ Reader::take_w_condition** (p. 1926), and the other variants of read() and take().

The general semantics of the `read()` operation is that the application only gets access to the corresponding data (i.e. a precise instance value); the data remains the responsibility of RTI Connext and can be read again.

The semantics of the `take()` operations is that the application takes full responsibility for the data; that data will no longer be available locally to RTI Connext. Consequently, it is possible to access the same information multiple times only if all previous accesses were read() operations, not take().

Each of these operations returns a collection of `Data` values and associated **DDS_SampleInfo** (p. 1344) objects. Each data value represents an atom of data information (i.e., a value for one instance). This collection may contain samples related to the same or different instances (identified by the key). Multiple samples can refer to the same instance if the settings of the **HISTORY** (p. 603) QoS allow for it.

These operations reset the read communication statuses; see **Changes in read communication status** (p. 536).

To return the memory back to the middleware, every read() or take() that retrieves a sequence of samples must be followed with a call to FooDataReader::return_loan (p. 1940).

See also

Interpretation of the SampleInfo (p. 1345)

7.19.2.1 Data access patterns

The application accesses data by means of the operations `read` or `take` on the **DDSDataReader** (p. 1556). These operations return an ordered collection of `DataSamples` consisting of a **DDS_SampleInfo** (p. 1344) part and a `Data` part.

The way RTI Connext builds the collection depends on QoS policies set on the **DDSDataReader** (p. 1556) and **DDSSubscriber** (p. 1860), as well as the `source_timestamp` of the samples, and the parameters passed to the `read()` / `take()` operations, namely:

- the desired sample states (any combination of **DDS_SampleStateKind** (p. 161))
- the desired view states (any combination of **DDS_ViewStateKind** (p. 163))

- the desired instance states (any combination of **DDS_InstanceStateKind** (p. 165))

The `read()` and `take()` operations are non-blocking and just deliver what is currently available that matches the specified states.

The `read_w_condition()` and `take_w_condition()` operations take a **DDSReadCondition** (p. 1842) object as a parameter instead of sample, view or instance states. The behaviour is that the samples returned will only be those for which the condition is **DDS_BOOLEAN_TRUE** (p. 513). These operations, in conjunction with **DDSReadCondition** (p. 1842) objects and a **DDSWaitSet** (p. 1897), allow performing waiting reads.

Once the data samples are available to the data readers, they can be read or taken by the application. The basic rule is that the application may do this in any order it wishes. This approach is very flexible and allows the application ultimate control.

To access data coherently, or in order, the **PRESENTATION** (p. 615) QoS must be set properly.

7.20 Subscribers

DDSSubscriber (p. 1860) entity and associated elements

Classes

- struct **DDS_SubscriberQos**

  *QoS policies supported by a **DDSSubscriber** (p. 1860) entity.*
- class **DDSSubscriberListener**

  *<<**interface**>> (p. 433) **DDSListener** (p. 1794) for status about a subscriber.*
- class **DDSSubscriberSeq**

  *Declares IDL `sequence` < **DDSSubscriber** (p. 1860) > .*
- class **DDSSubscriber**

  *<<**interface**>> (p. 433) A subscriber is the object responsible for actually receiving data from a subscription.*

Functions

- **DDS_Boolean DDS_SubscriberQos_equals** (const struct **DDS_SubscriberQos** *self, const struct **DDS_SubscriberQos** *other)

  *Compares two **DDS_SubscriberQos** (p. 1367) for equality.*
- **DDS_ReturnCode_t DDS_SubscriberQos::print** () const

  *Prints this **DDS_SubscriberQos** (p. 1367) to stdout.*
- **DDS_ReturnCode_t DDS_SubscriberQos::to_string** (char *string, **DDS_UnsignedLong** &string_size) const

  *Obtains a string representation of this **DDS_SubscriberQos** (p. 1367).*
- **DDS_ReturnCode_t DDS_SubscriberQos::to_string** (char *string, **DDS_UnsignedLong** &string_size, const **DDS_SubscriberQos** &base) const

  *Obtains a string representation of this **DDS_SubscriberQos** (p. 1367).*
- **DDS_ReturnCode_t DDS_SubscriberQos::to_string** (char *string, **DDS_UnsignedLong** &string_size, const **DDS_QosPrintFormat** &format) const

  *Obtains a string representation of this **DDS_SubscriberQos** (p. 1367).*

- **DDS_ReturnCode_t DDS_SubscriberQos::to_string** (char *string, **DDS_UnsignedLong** &string_size, const **DDS_SubscriberQos** &base, const **DDS_QosPrintFormat** &format) const

  *Obtains a string representation of this **DDS_SubscriberQos*** (p. *1367*).
- **DDS_ReturnCode_t DDS_SubscriberQos::to_string** (char *string, **DDS_UnsignedLong** &string_size, const **DDS_QosPrintAll_t** &) const

  *Obtains a string representation of this **DDS_SubscriberQos*** (p. *1367*).
- **DDS_ReturnCode_t DDS_SubscriberQos::to_string** (char *string, **DDS_UnsignedLong** &string_size, const **DDS_QosPrintAll_t** &, const **DDS_QosPrintFormat** &format) const

  *Obtains a string representation of this **DDS_SubscriberQos*** (p. *1367*).

Variables

- const struct **DDS_DataReaderQos DDS_DATAREADER_QOS_DEFAULT**

  Special value for creating data reader with default QoS.
- const struct **DDS_DataReaderQos DDS_DATAREADER_QOS_USE_TOPIC_QOS**

  *Special value for creating **DDSDataReader*** (p. *1556*) *with a combination of the default **DDS_DataReaderQos*** (p. *858*) *and the **DDS_TopicQos*** (p. *1397*).

7.20.1 Detailed Description

DDSSubscriber (p. 1860) entity and associated elements

7.20.2 Function Documentation

7.20.2.1 DDS_SubscriberQos_equals()

```
DDS_Boolean DDS_SubscriberQos_equals (
            const struct  DDS_SubscriberQos * self,
            const struct  DDS_SubscriberQos * other )
```

Compares two **DDS_SubscriberQos** (p. 1367) for equality.

Parameters

self	*<<**in**>> (p. 433)* This SubscriberQos.
other	*<<**in**>> (p. 433)* The other SubscriberQos to be compared with this SubscriberQos.

Returns

DDS_BOOLEAN_TRUE (p. 513) if the two Qos objects are equal or **DDS_BOOLEAN_FALSE** (p. 513) otherwise.

Referenced by **DDS_SubscriberQos::operator!=()**, and **DDS_SubscriberQos::operator==()**.

7.20.2.2 print()

```
DDS_ReturnCode_t DDS_SubscriberQos::print ( ) const  [inline]
```

Prints this **DDS_SubscriberQos** (p. 1367) to stdout.

Only the differences between this **DDS_SubscriberQos** (p. 1367) and the documented default are printed. If you wish to print everything regardless, see **DDS_SubscriberQos::to_string(char*, DDS_UnsignedLong&, const DDS_SubscriberQos&, const DDS_QosPrintFormat&) const** (p. 134). The output is formatted according to the default values for **DDS_QosPrintFormat** (p. 1285).

Returns

DDS_RETCODE_OK (p. 532) if no error was encountered.

7.20.2.3 to_string() [1/6]

```
DDS_ReturnCode_t DDS_SubscriberQos::to_string (
            char * string,
             DDS_UnsignedLong & string_size ) const  [inline]
```

Obtains a string representation of this **DDS_SubscriberQos** (p. 1367).

Only the differences between this **DDS_SubscriberQos** (p. 1367) and the documented default are printed to the string. If you wish to print everything regardless, see **DDS_SubscriberQos::to_string(char*, DDS_UnsignedLong&, const DDS_SubscriberQos&, const DDS_QosPrintFormat&) const** (p. 134). The output is formatted according to the default values for **DDS_QosPrintFormat** (p. 1285).

If the supplied buffer is NULL, the required length of the string is returned via the string_size parameter.

If the supplied buffer is not NULL, the string representation of the **DDS_SubscriberQos** (p. 1367) is written to the buffer.

Parameters

string	<<***out***>> *(p. 433)* The char buffer that will be used to store the string representation of this **DDS_SubscriberQos** (p. 1367). If NULL, this function will return the required length of this buffer through the string_size parameter.
string_size	<<***inout***>> *(p. 433)* Cannot be NULL. The size of the supplied buffer. If the supplied buffer is NULL, or if the supplied buffer is not large enough, this value will be updated with the required length of the buffer.

Returns

DDS_RETCODE_OK (p. 532) if no error was encountered.

See also

DDS_SubscriberQos::to_string(char*, DDS_UnsignedLong&, const DDS_SubscriberQos&, const DDS_↩ QosPrintFormat&) const (p. 134)

7.20.2.4 to_string() [2/6]

```
DDS_ReturnCode_t DDS_SubscriberQos::to_string (
            char * string,
             DDS_UnsignedLong & string_size,
            const  DDS_SubscriberQos & base ) const  [inline]
```

Obtains a string representation of this **DDS_SubscriberQos** (p. 1367).

This overload behaves the same as **DDS_SubscriberQos::to_string** (p. 132) but allows the caller to specify the **DDS↩ _SubscriberQos** (p. 1367), which is used as the base profile. Only the differences between the QoS and the base profile are included in the output string.

Parameters

string	<<***out***>> *(p. 433)* The char buffer that will be used to store the string representation of this **DDS_SubscriberQos** (p. 1367). If NULL, this function will return the required length of this buffer through the string_size parameter.
string_size	<<***inout***>> *(p. 433)* Cannot be NULL. The size of the supplied buffer. If the supplied buffer is NULL, or if the supplied buffer is not large enough, this value will be updated with the required length of the buffer.
base	<<***in***>> *(p. 433)* The **DDS_SubscriberQos** (p. 1367) to be used as the base QoS profile. Only the differences with respect to this base profile will be included in the output string.

Returns

DDS_RETCODE_OK (p. 532) if no error was encountered.

See also

DDS_SubscriberQos::to_string(char*, DDS_UnsignedLong&, const DDS_SubscriberQos&, const DDS_↩ QosPrintFormat&) const (p. 134)

References **DDS_QosPrintFormat_INITIALIZER**.

7.20.2.5 to_string() [3/6]

```
DDS_ReturnCode_t DDS_SubscriberQos::to_string (
            char * string,
             DDS_UnsignedLong & string_size,
            const  DDS_QosPrintFormat & format ) const  [inline]
```

Obtains a string representation of this **DDS_SubscriberQos** (p. 1367).

This overload behaves the same as **DDS_SubscriberQos::to_string** (p. 132) but allows the caller to specify the **DDS_QosPrintFormat** (p. 1285) which is used to format the output.

Parameters

string	*<<**out**>> (p. 433)* The char buffer that will be used to store the string representation of this **DDS_SubscriberQos** (p. 1367). If NULL, this function will return the required length of this buffer through the string_size parameter.
string_size	*<<**inout**>> (p. 433)* Cannot be NULL. The size of the supplied buffer. If the supplied buffer is NULL, or if the supplied buffer is not large enough, this value will be updated with the required length of the buffer.
format	*<<**in**>> (p. 433)* The **DDS_QosPrintFormat** (p. 1285) to be used to format the output string.

Returns

DDS_RETCODE_OK (p. 532) if no error was encountered.

See also

DDS_SubscriberQos::to_string(char*, DDS_UnsignedLong&, const DDS_SubscriberQos&, const DDS_QosPrintFormat&) const (p. 134)

References **DDS_RETCODE_ERROR**, and **DDS_RETCODE_OK**.

7.20.2.6 to_string() [4/6]

```
DDS_ReturnCode_t DDS_SubscriberQos::to_string (
            char * string,
             DDS_UnsignedLong & string_size,
            const  DDS_SubscriberQos & base,
            const  DDS_QosPrintFormat & format ) const  [inline]
```

Obtains a string representation of this **DDS_SubscriberQos** (p. 1367).

Only the differences between this **DDS_SubscriberQos** (p. 1367) and the **DDS_SubscriberQos** (p. 1367) supplied as the base are printed to the string.

If the supplied buffer is NULL, the required length of the string is returned via the string_size parameter.

If the supplied buffer is not NULL, the string representation of the **DDS_SubscriberQos** (p. 1367) is written to the buffer.

Parameters

string	<<***out***>> *(p. 433)* The char buffer that will be used to store the string representation of this **DDS_SubscriberQos** (p. 1367). If NULL, this function will return the required length of this buffer through the string_size parameter.
string_size	<<***inout***>> *(p. 433)* Cannot be NULL. The size of the supplied buffer. If the supplied buffer is NULL, or if the supplied buffer is not large enough, this value will be updated with the required length of the buffer.
base	<<***in***>> *(p. 433)* The **DDS_SubscriberQos** (p. 1367) to be used as the base QoS profile. Only the differences with respect to this base profile will be included in the output string.
format	<<***in***>> *(p. 433)* The **DDS_QosPrintFormat** (p. 1285) to be used to format the output string.

Returns

DDS_RETCODE_OK (p. 532) if no error was encountered.

7.20.2.7 to_string() [5/6]

```
DDS_ReturnCode_t DDS_SubscriberQos::to_string (
            char * string,
             DDS_UnsignedLong & string_size,
            const  DDS_QosPrintAll_t &  ) const  [inline]
```

Obtains a string representation of this **DDS_SubscriberQos** (p. 1367).

This overload behaves the same as **DDS_SubscriberQos::to_string** (p. 132) but prints the entire **DDS_Subscriber↩ Qos** (p. 1367) object. The only valid value for the argument of type **DDS_QosPrintAll_t** (p. 1285) is **DDS_QOS_↩ PRINT_ALL** (p. 169).

For example

```
SubscriberQos qos;
char *str = NULL;
DDS_UnsignedLong str_size = 0;
qos.to_string(str, str_size, QOS_PRINT_ALL);
```

The resultant string is formatted according to the default value for **DDS_QosPrintFormat** (p. 1285).

Parameters

string	<<***out***>> *(p. 433)* The char buffer that will be used to store the string representation of this **DDS_SubscriberQos** (p. 1367). If NULL, this function will return the required length of this buffer through the string_size parameter.
string_size	<<***inout***>> *(p. 433)* Cannot be NULL. The size of the supplied buffer. If the supplied buffer is NULL, or if the supplied buffer is not large enough, this value will be updated with the required length of the buffer.

Returns

DDS_RETCODE_OK (p. 532) if no error was encountered.

See also

DDS_SubscriberQos::to_string(char*, DDS_UnsignedLong&, const DDS_SubscriberQos&, const DDS_↩ QosPrintFormat&) const (p. 134)

References **DDS_QosPrintFormat_INITIALIZER**.

7.20.2.8 to_string() [6/6]

```
DDS_ReturnCode_t DDS_SubscriberQos::to_string (
            char * string,
             DDS_UnsignedLong & string_size,
            const  DDS_QosPrintAll_t & ,
            const  DDS_QosPrintFormat & format ) const  [inline]
```

Obtains a string representation of this **DDS_SubscriberQos** (p. 1367).

This overload behaves the same as the **DDS_SubscriberQos::to_string(char*, DDS_UnsignedLong&, const DDS↩ _QosPrintAll_t&) const** (p. 135) overload. The only difference is that it allows the caller to specify the **DDS_Qos↩ PrintFormat** (p. 1285) which is used to format the resultant string.

Parameters

string	<<***out***>> *(p. 433)* The char buffer that will be used to store the string representation of this **DDS_SubscriberQos** (p. 1367). If NULL, this function will return the required length of this buffer through the string_size parameter.
string_size	<<***inout***>> *(p. 433)* Cannot be NULL. The size of the supplied buffer. If the supplied buffer is NULL, or if the supplied buffer is not large enough, this value will be updated with the required length of the buffer.
format	<<***in***>> *(p. 433)* The **DDS_QosPrintFormat** (p. 1285) to be used to format the output string.

Returns

DDS_RETCODE_OK (p. 532) if no error was encountered.

See also

DDS_SubscriberQos::to_string(char*, DDS_UnsignedLong&, const DDS_SubscriberQos&, const DDS_↩ QosPrintFormat&) const (p. 134)

7.20.3 Variable Documentation

7.20.3.1 DDS_DATAREADER_QOS_DEFAULT

```
const struct DDS_DataReaderQos DDS_DATAREADER_QOS_DEFAULT [extern]
```

Special value for creating data reader with default QoS.

When used in **DDSSubscriber::create_datareader** (p. 1867), this special value is used to indicate that the **DDSDataReader** (p. 1556) should be created with the default **DDSDataReader** (p. 1556) QoS by means of the operation get_default_datareader_qos and using the resulting QoS to create the **DDSDataReader** (p. 1556).

When used in **DDSSubscriber::set_default_datareader_qos** (p. 1863), this special value is used to indicate that the default QoS should be reset back to the initial value that would be used if the **DDSSubscriber::set_default_datareader_qos** (p. 1863) operation had never been called.

When used in **DDSDataReader::set_qos** (p. 1574), this special value is used to indicate that the QoS of the **DDSDataReader** (p. 1556) should be changed to match the current default QoS set in the **DDSSubscriber** (p. 1860) that the **DDSDataReader** (p. 1556) belongs to.

Warning

This value is a constant and should never be modified. You cannot use this value to *get* the default QoS values for a DataReader; for this purpose, use **DDSDomainParticipant::get_default_datareader_qos** (p. 1628).

See also

DDSSubscriber::create_datareader (p. 1867)

DDSSubscriber::set_default_datareader_qos (p. 1863)

DDSDataReader::set_qos (p. 1574)

7.20.3.2 DDS_DATAREADER_QOS_USE_TOPIC_QOS

```
const struct DDS_DataReaderQos DDS_DATAREADER_QOS_USE_TOPIC_QOS [extern]
```

Special value for creating **DDSDataReader** (p. 1556) with a combination of the default **DDS_DataReaderQos** (p. 858) and the **DDS_TopicQos** (p. 1397).

The use of this value is equivalent to the application obtaining the default **DDS_DataReaderQos** (p. 858) and the **DDS_TopicQos** (p. 1397) (by means of the operation **DDSTopic::get_qos** (p. 1889)) and then combining these two QoS using the operation **DDSSubscriber::copy_from_topic_qos** (p. 1876) whereby any policy that is set on the **DDS_TopicQos** (p. 1397) "overrides" the corresponding policy on the default QoS. The resulting QoS is then applied to the creation of the **DDSDataReader** (p. 1556).

This value should only be used in **DDSSubscriber::create_datareader** (p. 1867).

See also

DDSSubscriber::create_datareader (p. 1867)

DDSSubscriber::get_default_datareader_qos (p. 1863)

DDSTopic::get_qos (p. 1889)

DDSSubscriber::copy_from_topic_qos (p. 1876)

7.21 DataReaders

DDSDataReader (p. 1556) entity and associated elements

Modules

- **Read Conditions**

  DDSReadCondition *(p. 1842) and associated elements*
- **Query Conditions**

  DDSQueryCondition *(p. 1841) and associated elements*
- **Topic Queries**

  DDSTopicQuery *(p. 1895) and associated elements.*

Classes

- struct **DDS_RequestedDeadlineMissedStatus**

  DDS_REQUESTED_DEADLINE_MISSED_STATUS *(p. 540)*
- struct **DDS_LivelinessChangedStatus**

  DDS_LIVELINESS_CHANGED_STATUS *(p. 542)*
- struct **DDS_RequestedIncompatibleQosStatus**

  DDS_REQUESTED_INCOMPATIBLE_QOS_STATUS *(p. 540)*
- struct **DDS_SampleLostStatus**

  DDS_SAMPLE_LOST_STATUS *(p. 541)*
- struct **DDS_SampleRejectedStatus**

  DDS_SAMPLE_REJECTED_STATUS *(p. 541)*
- struct **DDS_SubscriptionMatchedStatus**

  DDS_SUBSCRIPTION_MATCHED_STATUS *(p. 543)*
- struct **DDS_DataReaderCacheStatus**

  *<<**extension**>> (p. 432) The status of the reader's cache.*
- struct **DDS_DataReaderProtocolStatus**

  *<<**extension**>> (p. 432) The status of a reader's internal protocol related metrics, like the number of samples received, filtered, rejected; and status of wire protocol traffic.*
- struct **DDS_DataReaderQos**

  *QoS policies supported by a **DDSDataReader** (p. 1556) entity.*
- class **FooDataReader**

  *<<**interface**>> (p. 433) <<**generic**>> (p. 433) User data type-specific data reader.*
- class **DDSDataReaderSeq**

  *Declares IDL `sequence` < **DDSDataReader** (p. 1556) > .*
- class **DDSDataReaderListener**

  *<<**interface**>> (p. 433) **DDSListener** (p. 1794) for reader status.*
- class **DDSDataReader**

  *<<**interface**>> (p. 433) Allows the application to: (1) declare the data it wishes to receive (i.e. make a subscription) and (2) access the data received by the attached **DDSSubscriber** (p. 1860).*

Enumerations

- enum **DDS_SampleLostStatusKind** {
  DDS_NOT_LOST = 0 ,
  DDS_LOST_BY_WRITER = 1 ,
  DDS_LOST_BY_INSTANCES_LIMIT = 2 ,
  DDS_LOST_BY_REMOTE_WRITERS_PER_INSTANCE_LIMIT = 3 ,
  DDS_LOST_BY_INCOMPLETE_COHERENT_SET = 4 ,
  DDS_LOST_BY_LARGE_COHERENT_SET = 5 ,
  DDS_LOST_BY_SAMPLES_PER_REMOTE_WRITER_LIMIT = 6 ,
  DDS_LOST_BY_VIRTUAL_WRITERS_LIMIT = 7 ,
  DDS_LOST_BY_REMOTE_WRITERS_PER_SAMPLE_LIMIT = 8 ,
  DDS_LOST_BY_AVAILABILITY_WAITING_TIME = 9 ,
  DDS_LOST_BY_REMOTE_WRITER_SAMPLES_PER_VIRTUAL_QUEUE_LIMIT = 10 ,
  DDS_LOST_BY_OUT_OF_MEMORY = 11 ,
  DDS_LOST_BY_UNKNOWN_INSTANCE = 12 ,
  DDS_LOST_BY_DESERIALIZATION_FAILURE = 13 ,
  DDS_LOST_BY_DECODE_FAILURE = 14 ,
  DDS_LOST_BY_SAMPLES_PER_INSTANCE_LIMIT = 15 ,
  DDS_LOST_BY_SAMPLES_LIMIT = 16 }

  *<<**extension**>> (p. 432) Kinds of reasons why a sample was lost.*
- enum **DDS_SampleRejectedStatusKind** {
  DDS_NOT_REJECTED = 0 ,
  DDS_REJECTED_BY_INSTANCES_LIMIT = 1 ,
  DDS_REJECTED_BY_SAMPLES_LIMIT = 2 ,
  DDS_REJECTED_BY_SAMPLES_PER_INSTANCE_LIMIT = 3 ,
  DDS_REJECTED_BY_SAMPLES_PER_REMOTE_WRITER_LIMIT = 6 ,
  DDS_REJECTED_BY_REMOTE_WRITER_SAMPLES_PER_VIRTUAL_QUEUE_LIMIT = 9 ,
  DDS_REJECTED_BY_DECODE_FAILURE = 10 }

  Kinds of reasons for rejecting a sample.

Functions

- **DDS_Boolean DDS_DataReaderQos_equals** (const struct **DDS_DataReaderQos** *self, const struct **DDS_DataReaderQos** *other)

  *Compares two **DDS_DataReaderQos** (p. 858) for equality.*
- **DDS_ReturnCode_t DDS_DataReaderQos::print** () const

  *Prints this **DDS_DataReaderQos** (p. 858) to stdout.*
- **DDS_ReturnCode_t DDS_DataReaderQos::to_string** (char *string, **DDS_UnsignedLong** &string_size) const

  *Obtains a string representation of this **DDS_DataReaderQos** (p. 858).*
- **DDS_ReturnCode_t DDS_DataReaderQos::to_string** (char *string, **DDS_UnsignedLong** &string_size, const **DDS_DataReaderQos** &base) const

  *Obtains a string representation of this **DDS_DataReaderQos** (p. 858).*
- **DDS_ReturnCode_t DDS_DataReaderQos::to_string** (char *string, **DDS_UnsignedLong** &string_size, const **DDS_QosPrintFormat** &format) const

  *Obtains a string representation of this **DDS_DataReaderQos** (p. 858).*
- **DDS_ReturnCode_t DDS_DataReaderQos::to_string** (char *string, **DDS_UnsignedLong** &string_size, const **DDS_DataReaderQos** &base, const **DDS_QosPrintFormat** &format) const

  *Obtains a string representation of this **DDS_DataReaderQos** (p. 858).*

- **DDS_ReturnCode_t DDS_DataReaderQos::to_string** (char *string, **DDS_UnsignedLong** &string_size, const **DDS_QosPrintAll_t** &) const

  *Obtains a string representation of this **DDS_DataReaderQos** (p. 858).*
- **DDS_ReturnCode_t DDS_DataReaderQos::to_string** (char *string, **DDS_UnsignedLong** &string_size, const **DDS_QosPrintAll_t** &, const **DDS_QosPrintFormat** &format) const

  *Obtains a string representation of this **DDS_DataReaderQos** (p. 858).*

7.21.1 Detailed Description

DDSDataReader (p. 1556) entity and associated elements

7.21.2 Enumeration Type Documentation

7.21.2.1 DDS_SampleLostStatusKind

enum **DDS_SampleLostStatusKind**

<<***extension***>> *(p. 432)* Kinds of reasons why a sample was lost.

MONITOR-273 We assign an integer to the names in order to be compatible with the DDSMonitoring types.

Enumerator

DDS_NOT_LOST	The sample was not lost. This constant is an extension to the DDS standard.
DDS_LOST_BY_WRITER	A **DDSDataWriter** (p. 1589) removed the sample before being received by the **DDSDataReader** (p. 1556). This constant is an extension to the DDS standard.
DDS_LOST_BY_INSTANCES_LIMIT	A resource limit on the number of instances (**DDS_ResourceLimitsQosPolicy::max_instances** (p. 1311)) was reached. This constant is an extension to the DDS standard. **See also** **DDS_ResourceLimitsQosPolicy** (p. 1308)
DDS_LOST_BY_REMOTE_WRITERS_PER_↩ INSTANCE_LIMIT	A resource limit on the number of remote writers for a single instance from which a **DDSDataReader** (p. 1556) may read (**DDS_DataReaderResourceLimitsQos**↩ **Policy::max_remote_writers_per_instance** (p. 870)) was reached. This constant is an extension to the DDS standard. **See also** **DDS_DataReaderResourceLimitsQosPolicy** (p. 867)

Enumerator

DDS_LOST_BY_INCOMPLETE_COHERENT_SET	A sample is lost because it is part of an incomplete coherent set. An incomplete coherent set is a coherent set for which some of the samples are missing. For example, consider a **DDSDataWriter** (p. 1589) using **DDS_KEEP_LAST_HISTORY_QOS** (p. 604) with a depth of 1. The DataWriter publishes two samples of the same instance as part of a coherent set 'CS1'; the first sample of 'CS1' is replaced by a new sample before it can be successfully delivered to the **DDSDataReader** (p. 1556). In this case, the coherent set containing the two samples is considered incomplete. The new sample, by default, will not be provided to the application, and will be reported as LOST_BY_INCOMPLETE_COHERENT_SET. (You can change this default behavior by setting **DDS_PresentationQosPolicy::drop_incomplete_coherent_set** (p. 1253) to FALSE. If you do, the new sample will be provided to the application, but it will be marked as part of an incomplete coherent set in the **DDS_SampleInfo** (p. 1344) structure.) This constant is an extension to the DDS standard.
DDS_LOST_BY_LARGE_COHERENT_SET	A sample is lost because it is part of a large coherent set. A large coherent set is a coherent set that cannot fit all at once into the **DDSDataReader** (p. 1556) queue because resource limits are exceeded. For example, if **DDS_ResourceLimitsQosPolicy::max_samples_per_instance** (p. 1311) on the DataReader is 10 and the coherent set has 15 samples for a given instance, the coherent set is a large coherent set that will be considered incomplete. The resource limits that can lead to large coherent sets are: **DDS_ResourceLimitsQosPolicy::max_samples** (p. 1311), **DDS_ResourceLimitsQosPolicy::max_samples_per_instance** (p. 1311), **DDS_ResourceLimitsQosPolicy::max_instances** (p. 1311), and **DDS_DataReaderResourceLimitsQosPolicy::max_samples_per_remote_writer** (p. 870). This constant is an extension to the DDS standard.

Enumerator

DDS_LOST_BY_SAMPLES_PER_REMOTE_↩ WRITER_LIMIT	When using **DDS_BEST_EFFORT_RELIABILITY_QOS** (p. 633): a resource limit on the number of samples from a given remote writer that a **DDSDataReader** (p. 1556) may store (**DDS_DataReaderResourceLimitsQosPolicy↩ ::max_samples_per_remote_writer** (p. 870)) was reached. When using **DDS_RELIABLE_RELIABILITY_QOS** (p. 633), reaching **DDS_DataReaderResourceLimitsQos↩ Policy::max_samples_per_remote_writer** (p. 870) will trigger a rejection, not a loss, with reason **DDS_REJECTED_BY_SAMPLES_PER_REMOTE_↩ WRITER_LIMIT** (p. 145). This constant is an extension to the DDS standard. **See also** **DDS_DataReaderResourceLimitsQosPolicy** (p. 867)
DDS_LOST_BY_VIRTUAL_WRITERS_LIMIT	A resource limit on the number of virtual writers from which a **DDSDataReader** (p. 1556) may read (**DDS_DataReaderResourceLimitsQosPolicy::max↩ _remote_virtual_writers** (p. 878)) was reached. This constant is an extension to the DDS standard. **See also** **DDS_DataReaderResourceLimitsQosPolicy** (p. 867)
DDS_LOST_BY_REMOTE_WRITERS_PER_↩ SAMPLE_LIMIT	A resource limit on the number of remote writers per sample (**DDS_DataReaderResourceLimitsQos↩ Policy::max_remote_writers_per_sample** (p. 880)) was reached. This constant is an extension to the DDS standard. **See also** **DDS_DataReaderResourceLimitsQosPolicy** (p. 867)
DDS_LOST_BY_AVAILABILITY_WAITING_TIME	**DDS_AvailabilityQosPolicy::max_data_availability↩ _waiting_time** (p. 808) expired. This constant is an extension to the DDS standard. **See also** **DDS_AvailabilityQosPolicy** (p. 805)

Enumerator

DDS_LOST_BY_REMOTE_WRITER_SAMPLES_↩ PER_VIRTUAL_QUEUE_LIMIT	A resource limit on the number of samples published by a remote writer on behalf of a virtual writer that a **DDSDataReader** (p. 1556) may store was reached. (This field is currently not used.) This constant is an extension to the DDS standard. **See also** **DDS_DataReaderResourceLimitsQosPolicy** (p. 867)
DDS_LOST_BY_OUT_OF_MEMORY	A sample was lost because there was not enough memory to store the sample. This constant is an extension to the DDS standard. **See also** **DDS_DataReaderResourceLimitsQosPolicy** (p. 867)
DDS_LOST_BY_UNKNOWN_INSTANCE	A received sample was lost because it doesn't contain enough information for the reader to know what instance it is associated with. This constant is an extension to the DDS standard.
DDS_LOST_BY_DESERIALIZATION_FAILURE	A received sample was lost because it could not be deserialized. This constant is an extension to the DDS standard.
DDS_LOST_BY_DECODE_FAILURE	When using **DDS_BEST_EFFORT_RELIABILITY_QOS** (p. 633): A received sample was lost because it could not be decoded. When using **DDS_RELIABLE_RELIABILITY_QOS** (p. 633), the sample will be rejected, not lost, with reason **DDS_REJECTED_BY_DECODE_FAILURE** (p. 146). This constant is an extension to the DDS standard.
DDS_LOST_BY_SAMPLES_PER_INSTANCE_LIMIT	When using **DDS_BEST_EFFORT_RELIABILITY_QOS** (p. 633): A resource limit on the number of samples per instance (**DDS_ResourceLimitsQosPolicy::max_samples_↩ per_instance** (p. 1311)) was reached. When using **DDS_RELIABLE_RELIABILITY_QOS** (p. 633), reaching **DDS_ResourceLimitsQosPolicy::max_↩ samples_per_instance** (p. 1311) will trigger a rejection, not a loss, with reason **DDS_REJECTED_BY_↩ SAMPLES_PER_INSTANCE_LIMIT** (p. 145). This constant is an extension to the DDS standard. **See also** **DDS_ResourceLimitsQosPolicy** (p. 1308)

Enumerator

DDS_LOST_BY_SAMPLES_LIMIT	When using **DDS_BEST_EFFORT_RELIABILITY_QOS** (p. 633): A resource limit on the number of samples (**DDS_ResourceLimitsQosPolicy::max_samples** (p. 1311)) was reached. When using **DDS_RELIABLE_RELIABILITY_QOS** (p. 633), reaching **DDS_ResourceLimitsQosPolicy::max_samples** (p. 1311) will trigger a rejection, not a loss, with reason **DDS_REJECTED_BY_SAMPLES_LIMIT** (p. 144). This constant is an extension to the DDS standard. **See also** **DDS_ResourceLimitsQosPolicy** (p. 1308)

7.21.2.2 DDS_SampleRejectedStatusKind

enum **DDS_SampleRejectedStatusKind**

Kinds of reasons for rejecting a sample.

MONITOR-273 We assign an integer to the names in order to be compatible with the DDSMonitoring types.

Enumerator

DDS_NOT_REJECTED	The sample was not rejected.
DDS_REJECTED_BY_INSTANCES_LIMIT	Connext DDS does not reject samples based on instance limits (**DDS_ResourceLimitsQosPolicy::max_instances** (p. 1311)), so this value will never be used. **See also** **DDS_LOST_BY_INSTANCES_LIMIT** (p. 140) This value is not currently used by our middleware, but has been kept because it is part of the DDS specification.
DDS_REJECTED_BY_SAMPLES_LIMIT	When using **DDS_RELIABLE_RELIABILITY_QOS** (p. 633): A resource limit on the number of samples (**DDS_ResourceLimitsQosPolicy::max_samples** (p. 1311)) was reached. When using **DDS_BEST_EFFORT_RELIABILITY_QOS** (p. 633), reaching **DDS_ResourceLimitsQosPolicy::max_samples** (p. 1311) will trigger a loss, not a rejection, with reason **DDS_LOST_BY_SAMPLES_LIMIT** (p. 144). **See also**
	DDS_ResourceLimitsQosPolicy (p. 1308)

Enumerator

DDS_REJECTED_BY_SAMPLES_PER_INSTANCE_LIMIT	When using **DDS_RELIABLE_RELIABILITY_QOS** (p. 633): A resource limit on the number of samples per instance (**DDS_ResourceLimitsQosPolicy::max_samples_per_instance** (p. 1311)) was reached. When using **DDS_BEST_EFFORT_RELIABILITY_QOS** (p. 633), reaching **DDS_ResourceLimitsQosPolicy::max_samples_per_instance** (p. 1311) will trigger a loss, not a rejection, with reason **DDS_LOST_BY_SAMPLES_PER_INSTANCE_LIMIT** (p. 143). **See also** ResourceLimitsQosPolicy
DDS_REJECTED_BY_SAMPLES_PER_REMOTE_WRITER_LIMIT	When using **DDS_RELIABLE_RELIABILITY_QOS** (p. 633): a resource limit on the number of samples from a given remote writer that a **DDSDataReader** (p. 1556) may store (**DDS_DataReaderResourceLimitsQosPolicy::max_samples_per_remote_writer** (p. 870)) was reached. When using **DDS_BEST_EFFORT_RELIABILITY_QOS** (p. 633), reaching **DDS_DataReaderResourceLimitsQosPolicy::max_samples_per_remote_writer** (p. 870) will trigger a loss, not a rejection, with reason **DDS_LOST_BY_SAMPLES_PER_REMOTE_WRITER_LIMIT** (p. 142). This constant is an extension to the DDS standard. **See also** **DDS_DataReaderResourceLimitsQosPolicy** (p. 867)
DDS_REJECTED_BY_REMOTE_WRITER_SAMPLES_PER_VIRTUAL_QUEUE_LIMIT	A resource limit on the number of samples published by a remote writer on behalf of a virtual writer that a **DDSDataReader** (p. 1556) may store was reached. (This field is currently not used.) This constant is an extension to the DDS standard. **See also** **DDS_DataReaderResourceLimitsQosPolicy** (p. 867)

Enumerator

DDS_REJECTED_BY_DECODE_FAILURE	When using **DDS_RELIABLE_RELIABILITY_QOS** (p. 633): A received sample was rejected because it could not be decoded. When using **DDS_BEST_EFFORT_RELIABILITY_QOS** (p. 633), the sample will be lost, not rejected, with reason **DDS_LOST_BY_DECODE_FAILURE** (p. 143). If a sample was rejected for this reason and the **DDSDataWriter** (p. 1589) set **DDS_DataWriter↩ ProtocolQosPolicy::disable_inline_keyhash** (p. 895) to **DDS_BOOLEAN_TRUE** (p. 513), then **DDS_SampleRejectedStatus::last_instance_handle** (p. 1357) may not be correct if the sample was encrypted. This constant is an extension to the DDS standard.

7.21.3 Function Documentation

7.21.3.1 DDS_DataReaderQos_equals()

```
DDS_Boolean DDS_DataReaderQos_equals (
            const struct  DDS_DataReaderQos * self,
            const struct  DDS_DataReaderQos * other )
```

Compares two **DDS_DataReaderQos** (p. 858) for equality.

Parameters

self	<<***in***>> *(p. 433)* This DataReaderQos.
other	<<***in***>> *(p. 433)* The other DataReaderQos to be compared with this DataReaderQos.

Returns

DDS_BOOLEAN_TRUE (p. 513) if the two Qos objects are equal or **DDS_BOOLEAN_FALSE** (p. 513) otherwise.

Referenced by **DDS_DataReaderQos::operator!=()**, and **DDS_DataReaderQos::operator==()**.

7.21.3.2 print()

```
DDS_ReturnCode_t DDS_DataReaderQos::print ( ) const  [inline]
```

Prints this **DDS_DataReaderQos** (p. 858) to stdout.

Only the differences between this **DDS_DataReaderQos** (p. 858) and the documented default are printed. If you wish to print everything regardless, see **DDS_DataReaderQos::to_string(char*, DDS_UnsignedLong&, const DDS_DataReaderQos&, const DDS_QosPrintFormat&) const** (p. 149). The output is formatted according to the default values for **DDS_QosPrintFormat** (p. 1285).

Returns

DDS_RETCODE_OK (p. 532) if no error was encountered.

7.21.3.3 to_string() [1/6]

```
DDS_ReturnCode_t DDS_DataReaderQos::to_string (
            char * string,
             DDS_UnsignedLong & string_size ) const  [inline]
```

Obtains a string representation of this **DDS_DataReaderQos** (p. 858).

Only the differences between this **DDS_DataReaderQos** (p. 858) and the documented default are printed to the string. If you wish to print everything regardless, see **DDS_DataReaderQos::to_string(char*, DDS_UnsignedLong&, const DDS_DataReaderQos&, const DDS_QosPrintFormat&) const** (p. 149). The output is formatted according to the default values for **DDS_QosPrintFormat** (p. 1285).

If the supplied buffer is NULL, the required length of the string is returned via the string_size parameter.

If the supplied buffer is not NULL, the string representation of the **DDS_DataReaderQos** (p. 858) is written to the buffer.

Parameters

string	<<***out***>> *(p. 433)* The char buffer that will be used to store the string representation of this **DDS_DataReaderQos** (p. 858). If NULL, this function will return the required length of this buffer through the string_size parameter.
string_size	<<***inout***>> *(p. 433)* Cannot be NULL. The size of the supplied buffer. If the supplied buffer is NULL, or if the supplied buffer is not large enough, this value will be updated with the required length of the buffer.

Returns

DDS_RETCODE_OK (p. 532) if no error was encountered.

See also

DDS_DataReaderQos::to_string(char*, DDS_UnsignedLong&, const DDS_DataReaderQos&, const DDS_QosPrintFormat&) const (p. 149)

7.21.3.4 to_string() [2/6]

```
DDS_ReturnCode_t DDS_DataReaderQos::to_string (
            char * string,
             DDS_UnsignedLong & string_size,
            const  DDS_DataReaderQos & base ) const  [inline]
```

Obtains a string representation of this **DDS_DataReaderQos** (p. 858).

This overload behaves the same as **DDS_DataReaderQos::to_string** (p. 147) but allows the caller to specify the **DDS_DataReaderQos** (p. 858), which is used as the base profile. Only the differences between the QoS and the base profile are included in the output string.

Parameters

string	*<<**out**>> (p. 433)* The char buffer that will be used to store the string representation of this **DDS_DataReaderQos** (p. 858). If NULL, this function will return the required length of this buffer through the string_size parameter.
string_size	*<<**inout**>> (p. 433)* Cannot be NULL. The size of the supplied buffer. If the supplied buffer is NULL, or if the supplied buffer is not large enough, this value will be updated with the required length of the buffer.
base	*<<**in**>> (p. 433)* The **DDS_DataReaderQos** (p. 858) to be used as the base QoS profile. Only the differences with respect to this base profile will be included in the output string.

Returns

DDS_RETCODE_OK (p. 532) if no error was encountered.

See also

DDS_DataReaderQos::to_string(char*, DDS_UnsignedLong&, const DDS_DataReaderQos&, const DDS_QosPrintFormat&) const (p. 149)

References **DDS_QosPrintFormat_INITIALIZER**.

7.21.3.5 to_string() [3/6]

```
DDS_ReturnCode_t DDS_DataReaderQos::to_string (
            char * string,
             DDS_UnsignedLong & string_size,
            const  DDS_QosPrintFormat & format ) const  [inline]
```

Obtains a string representation of this **DDS_DataReaderQos** (p. 858).

This overload behaves the same as **DDS_DataReaderQos::to_string** (p. 147) but allows the caller to specify the **DDS_QosPrintFormat** (p. 1285) which is used to format the output.

Parameters

string	*<<**out**>> (p. 433)* The char buffer that will be used to store the string representation of this **DDS_DataReaderQos** (p. 858). If NULL, this function will return the required length of this buffer through the string_size parameter.
string_size	*<<**inout**>> (p. 433)* Cannot be NULL. The size of the supplied buffer. If the supplied buffer is NULL, or if the supplied buffer is not large enough, this value will be updated with the required length of the buffer.
format	*<<**in**>> (p. 433)* The **DDS_QosPrintFormat** (p. 1285) to be used to format the output string.

Returns

DDS_RETCODE_OK (p. 532) if no error was encountered.

See also

DDS_DataReaderQos::to_string(char*, DDS_UnsignedLong&, const DDS_DataReaderQos&, const DDS_QosPrintFormat&) const (p. 149)

References **DDS_RETCODE_ERROR**, and **DDS_RETCODE_OK**.

7.21.3.6 to_string() [4/6]

```
DDS_ReturnCode_t DDS_DataReaderQos::to_string (
            char * string,
             DDS_UnsignedLong & string_size,
            const  DDS_DataReaderQos & base,
            const  DDS_QosPrintFormat & format ) const  [inline]
```

Obtains a string representation of this **DDS_DataReaderQos** (p. 858).

Only the differences between this **DDS_DataReaderQos** (p. 858) and the **DDS_DataReaderQos** (p. 858) supplied as the base are printed to the string.

If the supplied buffer is NULL, the required length of the string is returned via the string_size parameter.

If the supplied buffer is not NULL, the string representation of the **DDS_DataReaderQos** (p. 858) is written to the buffer.

Parameters

string	*<<**out**>> (p. 433)* The char buffer that will be used to store the string representation of this **DDS_DataReaderQos** (p. 858). If NULL, this function will return the required length of this buffer through the string_size parameter.
string_size	*<<**inout**>> (p. 433)* Cannot be NULL. The size of the supplied buffer. If the supplied buffer is NULL, or if the supplied buffer is not large enough, this value will be updated with the required length of the buffer.
base	*<<**in**>> (p. 433)* The **DDS_DataReaderQos** (p. 858) to be used as the base QoS profile. Only the differences with respect to this base profile will be included in the output string.
format	*<<**in**>> (p. 433)* The **DDS_QosPrintFormat** (p. 1285) to be used to format the output string.

Returns

DDS_RETCODE_OK (p. 532) if no error was encountered.

7.21.3.7 to_string() [5/6]

```
DDS_ReturnCode_t DDS_DataReaderQos::to_string (
            char * string,
             DDS_UnsignedLong & string_size,
            const  DDS_QosPrintAll_t &  ) const  [inline]
```

Obtains a string representation of this **DDS_DataReaderQos** (p. 858).

This overload behaves the same as **DDS_DataReaderQos::to_string** (p. 147) but prints the entire **DDS_DataReader↩ Qos** (p. 858) object. The only valid value for the argument of type **DDS_QosPrintAll_t** (p. 1285) is **DDS_QOS_PRINT↩ _ALL** (p. 169).

For example
```
DataReaderQos qos;
char *str = NULL;
DDS_UnsignedLong str_size = 0;
qos.to_string(str, str_size, QOS_PRINT_ALL);
```

The resultant string is formatted according to the default value for **DDS_QosPrintFormat** (p. 1285).

Parameters

string	<<***out***>> *(p. 433)* The char buffer that will be used to store the string representation of this **DDS_DataReaderQos** (p. 858). If NULL, this function will return the required length of this buffer through the string_size parameter.
string_size	<<***inout***>> *(p. 433)* Cannot be NULL. The size of the supplied buffer. If the supplied buffer is NULL, or if the supplied buffer is not large enough, this value will be updated with the required length of the buffer.

Returns

DDS_RETCODE_OK (p. 532) if no error was encountered.

See also

DDS_DataReaderQos::to_string(char∗, DDS_UnsignedLong&, const DDS_DataReaderQos&, const DDS↩ _QosPrintFormat&) const (p. 149)

References **DDS_QosPrintFormat_INITIALIZER**.

7.21.3.8 to_string() [6/6]

```
DDS_ReturnCode_t DDS_DataReaderQos::to_string (
            char * string,
             DDS_UnsignedLong & string_size,
            const  DDS_QosPrintAll_t & ,
            const  DDS_QosPrintFormat & format ) const  [inline]
```

Obtains a string representation of this **DDS_DataReaderQos** (p. 858).

This overload behaves the same as the **DDS_DataReaderQos::to_string(char∗, DDS_UnsignedLong&, const DDS_QosPrintAll_t&) const** (p. 150) overload. The only difference is that it allows the caller to specify the **DDS↩ _QosPrintFormat** (p. 1285) which is used to format the resultant string.

Parameters

string	*<<**out**>> (p. 433)* The char buffer that will be used to store the string representation of this **DDS_DataReaderQos** (p. 858). If NULL, this function will return the required length of this buffer through the string_size parameter.
string_size	*<<**inout**>> (p. 433)* Cannot be NULL. The size of the supplied buffer. If the supplied buffer is NULL, or if the supplied buffer is not large enough, this value will be updated with the required length of the buffer.
format	*<<**in**>> (p. 433)* The **DDS_QosPrintFormat** (p. 1285) to be used to format the output string.

Returns

DDS_RETCODE_OK (p. 532) if no error was encountered.

See also

DDS_DataReaderQos::to_string(char∗, DDS_UnsignedLong&, const DDS_DataReaderQos&, const DDS↩ _QosPrintFormat&) const (p. 149)

7.22 Read Conditions

DDSReadCondition (p. 1842) and associated elements

Classes

- struct **DDS_ReadConditionParams**

  *<<**extension**>> (p. 432) Input parameters for **DDSDataReader::create_readcondition_w_params** (p. 1561)*
- class **DDSReadCondition**

  *<<**interface**>> (p. 433) Conditions specifically dedicated to read operations and attached to one **DDSDataReader** (p. 1556).*

7.22.1 Detailed Description

DDSReadCondition (p. 1842) and associated elements

7.23 Query Conditions

DDSQueryCondition (p. 1841) and associated elements

Classes

- struct **DDS_QueryConditionParams**

  <<***extension***>> *(p. 432) Input parameters for* ***DDSDataReader::create_querycondition_w_params*** *(p. 1562)*
- class **DDSQueryCondition**

  <<***interface***>> *(p. 433) These are specialised* ***DDSReadCondition*** *(p. 1842) objects that allow the application to also specify a filter on the locally available data.*

7.23.1 Detailed Description

DDSQueryCondition (p. 1841) and associated elements

7.24 Data Samples

DDS_SampleInfo (p. 1344), **DDS_SampleStateKind** (p. 161), **DDS_ViewStateKind** (p. 163), **DDS_InstanceStateKind** (p. 165) and associated elements

Modules

- **Sample States**

  DDS_SampleStateKind *(p. 161) and associated elements*
- **View States**

  DDS_ViewStateKind *(p. 163) and associated elements*
- **Instance States**

  DDS_InstanceStateKind *(p. 165) and associated elements*
- **Stream Kinds**

  DDS_StreamKind *(p. 167) and associated elements*

Classes

- struct **DDS_CoherentSetInfo_t**

  <<***extension***>> (p. *432) Type definition for a coherent set info.*
- struct **DDS_SampleInfo**

  Information that accompanies each sample that is `read` *or* `taken`*.*
- struct **DDS_SampleInfoSeq**

  Declares IDL `sequence` < ***DDS_SampleInfo*** (p. *1344)* > .

Functions

- **DDS_Boolean** **DDS_CoherentSetInfo_equals** (const struct **DDS_CoherentSetInfo_t** ∗self, const struct **DDS_CoherentSetInfo_t** ∗other)

  Compares this CoherentSetInfo with another CoherentSetInfo for equality.
- void **DDS_CoherentSetInfo_copy** (struct **DDS_CoherentSetInfo_t** ∗self, const struct **DDS_CoherentSet↩ Info_t** ∗other)

  Copies another CoherentSetInfo into this CoherentSetInfo.
- void **DDS_SampleInfo_get_sample_identity** (const struct **DDS_SampleInfo** ∗self, struct **DDS_Sample↩ Identity_t** ∗identity)

  <<***extension***>> (p. *432) Retrieves the identity of the sample*
- void **DDS_SampleInfo_get_related_sample_identity** (const struct **DDS_SampleInfo** ∗self, struct **DDS_↩ SampleIdentity_t** ∗related_identity)

  <<***extension***>> (p. *432) Retrieves the identity of a sample related to this one*

7.24.1 Detailed Description

DDS_SampleInfo (p. 1344), **DDS_SampleStateKind** (p. 161), **DDS_ViewStateKind** (p. 163), **DDS_InstanceState↩ Kind** (p. 165) and associated elements

7.24.2 Function Documentation

7.24.2.1 DDS_CoherentSetInfo_equals()

```
DDS_Boolean DDS_CoherentSetInfo_equals (
            const struct  DDS_CoherentSetInfo_t * self,
            const struct  DDS_CoherentSetInfo_t * other )
```

Compares this CoherentSetInfo with another CoherentSetInfo for equality.

Parameters

self	<<***in***>> (p. *433)* This CoherentSetInfo. Cannot be NULL.
other	<<***in***>> (p. *433)* The other CoherentSetInfo to be compared with this GUID. Cannot be NULL.

Returns

DDS_BOOLEAN_TRUE (p. 513) if the two CoherentSetInfos have equal values, or **DDS_BOOLEAN_FALSE** (p. 513) otherwise.

7.24.2.2 DDS_CoherentSetInfo_copy()

```
void DDS_CoherentSetInfo_copy (
            struct  DDS_CoherentSetInfo_t * self,
            const struct  DDS_CoherentSetInfo_t * other )
```

Copies another CoherentSetInfo into this CoherentSetInfo.

Parameters

self	<<***in***>> *(p. 433)* This CoherentSetInfo. Cannot be NULL.
other	<<***in***>> *(p. 433)* The other CoherentSetInfo to be copied.

7.24.2.3 DDS_SampleInfo_get_sample_identity()

```
void DDS_SampleInfo_get_sample_identity (
            const struct  DDS_SampleInfo * self,
            struct  DDS_SampleIdentity_t * identity )
```

<<***extension***>> *(p. 432)* Retrieves the identity of the sample

The identity is composed of the **DDS_SampleInfo::original_publication_virtual_guid** (p. 1352) and the **DDS_SampleInfo::original_publication_virtual_sequence_number** (p. 1352)

See also

DDS_SampleInfo_get_related_sample_identity (p. 154)

7.24.2.4 DDS_SampleInfo_get_related_sample_identity()

```
void DDS_SampleInfo_get_related_sample_identity (
            const struct  DDS_SampleInfo * self,
            struct  DDS_SampleIdentity_t * related_identity )
```

<<***extension***>> *(p. 432)* Retrieves the identity of a sample related to this one

A sample can be logically related to another sample, when the **DDSDataWriter** (p. 1589) wrote it using **DDS_WriteParams_t::related_sample_identity** (p. 1520). By default, a sample does not have a related sample, and this operation returns **DDS_UNKNOWN_SAMPLE_IDENTITY** (p. 676).

The related identity is composed of the **DDS_SampleInfo::related_original_publication_virtual_guid** (p. 1352) and the **DDS_SampleInfo::related_original_publication_virtual_sequence_number** (p. 1353)

See also

DDS_WriteParams_t::related_sample_identity (p. 1520)

7.25 Topic Queries

DDSTopicQuery (p. 1895) and associated elements.

Classes

- struct **DDS_TopicQuerySelection**

  <<***extension***>> (p. 432) *Specifies the data query that defines a* ***DDSTopicQuery*** (p. 1895)
- struct **DDS_TopicQueryData**

  <<***extension***>> (p. 432) *Provides information about a* ***DDSTopicQuery*** (p. 1895)
- class **DDSTopicQuery**

  <<***extension***>> (p. 432) *Allows a* ***DDSDataReader*** (p. 1556) *to query the sample cache of its matching* ***DDSDataWriter*** (p. 1589).
- class **DDSTopicQueryHelper**

  Helpers to provide utility operations related to ***DDSTopicQuery*** (p. 1895).

Typedefs

- typedef struct **DDS_TopicQuerySelection** **DDS_TopicQuerySelection**

  <<***extension***>> (p. 432) *Specifies the data query that defines a* ***DDSTopicQuery*** (p. 1895)
- typedef struct **DDS_TopicQueryData** **DDS_TopicQueryData**

  <<***extension***>> (p. 432) *Provides information about a* ***DDSTopicQuery*** (p. 1895)

Enumerations

- enum **DDS_TopicQuerySelectionKind** {
  DDS_TOPIC_QUERY_SELECTION_KIND_HISTORY_SNAPSHOT ,
  DDS_TOPIC_QUERY_SELECTION_KIND_CONTINUOUS }

  Kinds of TopicQuerySelection.

Variables

- const struct **DDS_TopicQuerySelection** **DDS_TOPIC_QUERY_SELECTION_USE_READER_CONTENT_FILTER**

  Special value for creating a ***DDSTopicQuery*** (p. 1895) *that applies the same filter as the DataReader's* ***DDSContentFilteredTopic*** (p. 1551).
- const struct **DDS_TopicQuerySelection** **DDS_TOPIC_QUERY_SELECTION_SELECT_ALL**

  Special value for creating a ***DDSTopicQuery*** (p. 1895) *that selects all the samples in a* ***DDSDataWriter*** (p. 1589) *cache.*

7.25.1 Detailed Description

DDSTopicQuery (p. 1895) and associated elements.

TopicQueries allow a **DDSDataReader** (p. 1556) to query the sample cache of its matching **DDSDataWriter** (p. 1589). A user can create a **DDSTopicQuery** (p. 1895) with the **DDSDataReader::create_topic_query** (p. 1577) API. When a DataReader
creates a TopicQuery, DDS will propagate it to other DomainParticipants and their DataWriters. When a DataWriter matching with the DataReader that created the TopicQuery receives it, it will send the cached samples that pass the TopicQuery's filter.

To configure how to dispatch a TopicQuery, the **DDS_DataWriterQos** (p. 909) includes the **DDS_TopicQuery↩ DispatchQosPolicy** (p. 1404) policy. By default, a DataWriter ignores TopicQueries unless they are explicitly enabled using this policy.

The delivery of TopicQuery samples occurs in a separate RTPS channel. This allows DataReaders to receive Topic↩ Query samples and live samples in parallel. This is a key difference with respect to the Durability QoS policy.

Late-joining DataWriters will also discover existing TopicQueries. To delete a TopicQuery you must use **DDSData↩ Reader::delete_topic_query** (p. 1578).

After deleting a TopicQuery, new DataWriters won't discover it and existing DataWriters currently publishing cached samples may stop before delivering all of them.

By default, a TopicQuery queries the samples that were in the DataWriter's queue at the time the DataWriter received the TopicQuery. These TopicQueries are snapshot TopicQueries. However, a TopicQuery can be created in "continuous" mode; in this case, a DataWriter will continue delivering samples that pass a continuous TopicQuery filter until the DataReader application explicitly deletes it. These TopicQueries are continuous TopicQueries.

The samples received in response to a TopicQuery are stored in the associated DataReader's cache. Any of the read/take operations can retrieve TopicQuery samples. The field **DDS_SampleInfo::topic_query_guid** (p. 1354) associates each sample to its TopicQuery. If the read sample is not in response to a TopicQuery then this field will be **DDS_GUID_UNKNOWN** (p. 527). Note that the same data may be received several times, depending on how many TopicQueries the DataReader creates and their TopicQuerySelection.

You can choose to read or take only TopicQuery samples, only live samples, or both. To support this ReadConditions and QueryConditions provide the **DDSDataReader::create_querycondition_w_params** (p. 1562) and **DDSDataReader↩ ::create_readcondition_w_params** (p. 1561) APIs.

Each TopicQuery is identified by a GUID that can be accessed using the **DDSTopicQuery::get_guid** (p. 1895) method.

7.25.2 Snapshot TopicQueries

A snapshot TopicQuery queries the samples that are in a DataWriter's queue when the DataWriter receives the Topic↩ Query.

When a DataWriter receives a snapshot TopicQuery, it registers the last sequence number available in the DataWriter queue (known as the cut-off SN). Then, the DataWriter applies the TopicQuery expression to all the samples up to the cut-off SN. If a sample with a SN smaller than or equal to the cut-off SN passes the filter expression, it will be included in the TopicQuery response.

When a DataWriter responds to a snapshot TopicQuery, it doesn't create any copies of the samples that are part of the response. As a result, a sample that passes the TopicQuery expression and should have been included in the response may be removed from the DataWriter queue before it reaches the DataReader that issued the TopicQuery. This can result in incomplete TopicQueries, which may or may not be necessary for your application to detect.

For instance, your application may issue a TopicQuery on a keyed topic using a filter on key-only fields and expect to receive the latest value for an instance that passes the filter. However, if the latest value for the instance is replaced with a newer value before the sample is delivered to the DataReader, it is important to detect this situation and re-issue the TopicQuery.

On the other hand, some applications may not care about which samples are returned in response to a TopicQuery, and are only interested in a snapshot of the DataWriter's queue. In such cases, the concept of incomplete TopicQueries can be ignored.

The application can detect incomplete TopicQueries by checking the **DDS_INCOMPLETE_SNAPSHOT_TOPIC_↩ QUERY** (p. 674) bit in the **DDS_SampleInfo::flag** (p. 1353) field of the last sample received for a snapshot TopicQuery.

To reduce the probability of receiving incomplete responses, DataWriters configured with **DDS_KEEP_LAST_↩ HISTORY_QOS** (p. 604) recalculate the cut-off SN for a snapshot TopicQuery if a sample is replaced in the DataWriter queue due to KEEP_LAST replacement before the complete TopicQuery response is published.

Snapshot TopicQueries are not automatically deleted once the DataWriters that respond to the TopicQuery have finished delivering the samples included in the response. It is the responsibility of the user to delete the TopicQuery when it is no longer needed. To detect the end of a snapshot TopicQuery response, check for the **DDS_INTERMEDIATE_TOPIC↩ _QUERY_SAMPLE** (p. 674) flag in the **DDS_SampleInfo::flag** (p. 1353) field of the samples included in the response. For more information on how to use this flag, see **Reading TopicQuery Samples** (p. 157).

7.25.3 Reading TopicQuery Samples

A TopicQuery response is the set of samples published by a DataWriter in response to a TopicQuery. Because multiple DataWriters may respond to a TopicQuery, a DataReader may receive multiple responses for a single TopicQuery.

Responses can be proxied by Routing Service DataWriters by setting <topic_query_proxy>/<mode> to PROPAGATION. These DataWriters forward a TopicQuery response, but they are not considered to be responding to the Topic↩ Query.

Samples that a DataReader receives in response to a TopicQuery have a value for **DDS_SampleInfo::topic_query_↩ guid** (p. 1354) different than **DDS_GUID_UNKNOWN** (p. 527).

In addition, the virtual GUID of the DataWriter(s) that responded to a TopicQuery is set in **DDS_SampleInfo::source↩ _guid** (p. 1353).

Intermediate samples, not the final sample, have the flag **DDS_INTERMEDIATE_TOPIC_QUERY_SAMPLE** (p. 674) set in **DDS_SampleInfo::flag** (p. 1353) for a given response to a snapshot TopicQuery.

If two DataWriters respond to a snapshot TopicQuery, you should expect two final samples, one per unique **DDS↩ _SampleInfo::source_guid** (p. 1353) responding to the TopicQuery, assuming that at least one sample passes the TopicQuery filter for each DataWriter.

The **DDS_SampleInfo::flag** (p. 1353) field of the last sample received for a snapshot TopicQuery may have the **DDS↩ _INCOMPLETE_SNAPSHOT_TOPIC_QUERY** (p. 674) bit set if the TopicQuery response is incomplete, which may or may not be necessary for your application to detect. See **Snapshot TopicQueries** (p. 156) for more information.

7.25.4 Debugging Topic Queries

There are a number of ways in which to gain more insight into what is happening in an application that is creating Topic Queries.

7.25.4.1 The Built-in ServiceRequest DataReader

TopicQueries are communicated to publishing applications through a built-in **DDS_ServiceRequest** (p. 1359) channel. The ServiceRequest channel is designed to be generic so that it can be used for many different purposes, one of which is TopicQueries.

When a DataReader creates a TopicQuery, a **DDS_ServiceRequest** (p. 1359) message is sent containing the TopicQuery information. Just as there are built-in DataReaders for **DDS_ParticipantBuiltinTopicData** (p. 1229), **DDS_SubscriptionBuiltinTopicData** (p. 1370), and **DDS_PublicationBuiltinTopicData** (p. 1264), there is a fourth built-in DataReader for ServiceRequests.

The new built-in DataReader can be retrieved using the built-in subscriber and **DDSSubscriber::lookup_datareader** (p. 1872). The topic name is **DDS_SERVICE_REQUEST_TOPIC_NAME** (p. 499). Installing a listener with the **DDSDataReaderListener::on_data_available** (p. 1585) callback implemented will allow a publishing application to be notified whenever a TopicQuery has been received from a subscribing application.

The **DDS_ServiceRequest::service_id** (p. 1360) of a **DDS_ServiceRequest** (p. 1359) corresponding to a **DDSTopicQuery** (p. 1895) will be **DDS_TOPIC_QUERY_SERVICE_REQUEST_ID** (p. 498) and the **DDS_ServiceRequest::instance_id** (p. 1360) will be equal to the GUID of the **DDSTopicQuery** (p. 1895) (see **DDSTopicQuery::get_guid** (p. 1895)).

The **DDS_ServiceRequest::request_body** (p. 1360) is a sequence of bytes containing more information about the TopcQuery. This information can be retrieved using the **DDSTopicQueryHelper::topic_query_data_from_service_request** (p. 1896) function. The resulting **DDS_TopicQueryData** (p. 1402) contains the **DDS_TopicQuerySelection** (p. 1406) that the **DDSTopicQuery** (p. 1895) was created with, the GUID of the original DataReader that created the TopicQuery, and the topic name of that DataReader. Note: When TopicQueries are propagated through one or more Routing Services, the last DataReader that issued the TopicQuery will be a Routing Service DataReader. The **DDS_TopicQueryData::original_related_reader_guid** (p. 1403), however, will be that of the first DataReader to have created the TopicQuery.

7.25.4.2 The on_service_request_accepted DataWriter Listener Callback

It is possible that a **DDS_ServiceRequest** (p. 1359) for a **DDSTopicQuery** (p. 1895) is received but is not immediately dispatched to a DataWriter. This can happen, for example, if a DataWriter was not matching with a DataReader at the time that the TopicQuery was received by the publishing application. The **DDSDataWriterListener::on_service_request_accepted** (p. 1618) callback notifies a DataWriter when a ServiceRequest has been dispatched to that DataWriter. The **DDS_ServiceRequestAcceptedStatus** (p. 1361) provides information about how many ServiceRequests have been accepted by the DataWriter since the last time that the status was read. The status also includes the **DDS_ServiceRequestAcceptedStatus::last_request_handle** (p. 1362), which is the **DDS_InstanceHandle_t** (p. 76) of the last ServiceRequest that was accepted. This instance handle can be used to read samples per instance from the built-in ServiceRequest DataReader and correlate which ServiceRequests have been dispatched to which DataWriters.

7.25.5 Typedef Documentation

7.25.5.1 DDS_TopicQuerySelection

```
typedef struct DDS_TopicQuerySelection DDS_TopicQuerySelection
```

<<***extension***>> *(p. 432)* Specifies the data query that defines a **DDSTopicQuery** (p. 1895)

The query format is similar to the expression and parameters of a **DDSQueryCondition** (p. 1841) or a **DDSContentFilteredTopic** (p. 1551), as described in **DDSDomainParticipant::create_contentfilteredtopic_with_filter** (p. 1654).

There are two special queries:

- **DDS_TOPIC_QUERY_SELECTION_SELECT_ALL** (p. 160)
- **DDS_TOPIC_QUERY_SELECTION_USE_READER_CONTENT_FILTER** (p. 160)

See also

Queries and Filters Syntax (p. 184)

7.25.5.2 DDS_TopicQueryData

```
typedef struct DDS_TopicQueryData DDS_TopicQueryData
```

<<***extension***>> *(p. 432)* Provides information about a **DDSTopicQuery** (p. 1895)

Contains the information about a TopicQuery that can be retrieved using **DDSTopicQueryHelper::topic_query_data_from_service_request** (p. 1896).

See also

DDSTopicQueryHelper::topic_query_data_from_service_request (p. 1896)

7.25.6 Enumeration Type Documentation

7.25.6.1 DDS_TopicQuerySelectionKind

```
enum DDS_TopicQuerySelectionKind
```

Kinds of TopicQuerySelection.

See also

DDS_TopicQuerySelection (p. 1406)

Enumerator

DDS_TOPIC_QUERY_SELECTION_KIND_↩ HISTORY_SNAPSHOT	Indicates that the **DDSTopicQuery** (p. 1895) may only select samples that were in the DataWriter cache upon reception. **[default]**
DDS_TOPIC_QUERY_SELECTION_KIND_↩ CONTINUOUS	Indicates that the **DDSTopicQuery** (p. 1895) may continue selecting samples published after its reception. The subscribing application must explicitly delete the TopicQuery (see **DDSDataReader::delete_topic_query** (p. 1578)) to signal the DataWriters to stop publishing samples for it.

7.25.7 Variable Documentation

7.25.7.1 DDS_TOPIC_QUERY_SELECTION_USE_READER_CONTENT_FILTER

```
const struct DDS_TopicQuerySelection DDS_TOPIC_QUERY_SELECTION_USE_READER_CONTENT_FILTER [extern]
```

Special value for creating a **DDSTopicQuery** (p. 1895) that applies the same filter as the DataReader's **DDSContent↩ FilteredTopic** (p. 1551).

If the **DDSDataReader** (p. 1556) that creates the **DDSTopicQuery** (p. 1895) uses a **DDSContentFilteredTopic** (p. 1551), this **DDS_TopicQuerySelection** (p. 1406) value indicates that the TopicQuery should use the same filter.

Otherwise it behaves as **DDS_TOPIC_QUERY_SELECTION_SELECT_ALL** (p. 160).

7.25.7.2 DDS_TOPIC_QUERY_SELECTION_SELECT_ALL

```
const struct DDS_TopicQuerySelection DDS_TOPIC_QUERY_SELECTION_SELECT_ALL [extern]
```

Special value for creating a **DDSTopicQuery** (p. 1895) that selects all the samples in a **DDSDataWriter** (p. 1589) cache.

7.26 Sample States

DDS_SampleStateKind (p. 161) and associated elements

Typedefs

- typedef **DDS_UnsignedLong DDS_SampleStateMask**

  *A bit-mask (list) of sample states, i.e. **DDS_SampleStateKind** (p. 161).*

Enumerations

- enum **DDS_SampleStateKind** {
  DDS_READ_SAMPLE_STATE = 0x0001 << 0 ,
  DDS_NOT_READ_SAMPLE_STATE = 0x0001 << 1 }

  Indicates whether or not a sample has ever been read.

Variables

- const **DDS_SampleStateMask** **DDS_ANY_SAMPLE_STATE**

  *Any sample state **DDS_READ_SAMPLE_STATE** (p. 162) | **DDS_NOT_READ_SAMPLE_STATE** (p. 162).*

7.26.1 Detailed Description

DDS_SampleStateKind (p. 161) and associated elements

7.26.2 Typedef Documentation

7.26.2.1 DDS_SampleStateMask

`typedef` **`DDS_UnsignedLong DDS_SampleStateMask`**

A bit-mask (list) of sample states, i.e. **DDS_SampleStateKind** (p. 161).

7.26.3 Enumeration Type Documentation

7.26.3.1 DDS_SampleStateKind

`enum` **`DDS_SampleStateKind`**

Indicates whether or not a sample has ever been read.

For each sample received, the middleware internally maintains a sample_state relative to each **DDSDataReader** (p. 1556). The sample state can be either:

- **DDS_READ_SAMPLE_STATE** (p. 162) indicates that the **DDSDataReader** (p. 1556) has already accessed that sample by means of a read or take operation.
- **DDS_NOT_READ_SAMPLE_STATE** (p. 162) indicates that the **DDSDataReader** (p. 1556) has not accessed that sample before.

The sample state will, in general, be different for each sample in the collection returned by read or take.

Enumerator

DDS_READ_SAMPLE_STATE	Sample has been read.
DDS_NOT_READ_SAMPLE_STATE	Sample has not been read.

7.26.4 Variable Documentation

7.26.4.1 DDS_ANY_SAMPLE_STATE

```
const DDS_SampleStateMask DDS_ANY_SAMPLE_STATE [extern]
```

Any sample state **DDS_READ_SAMPLE_STATE** (p. 162) | **DDS_NOT_READ_SAMPLE_STATE** (p. 162).

Examples

HelloWorld_subscriber.cxx.

7.27 View States

DDS_ViewStateKind (p. 163) and associated elements

Typedefs

- typedef **DDS_UnsignedLong DDS_ViewStateMask**
  *A bit-mask (list) of view states, i.e. **DDS_ViewStateKind** (p. 163).*

Enumerations

- enum **DDS_ViewStateKind** {
  DDS_NEW_VIEW_STATE = 0x0001 << 0 ,
  DDS_NOT_NEW_VIEW_STATE = 0x0001 << 1 }
  Indicates whether or not an instance is new.

Variables

- const **DDS_ViewStateMask DDS_ANY_VIEW_STATE**
  *Any view state **DDS_NEW_VIEW_STATE** (p. 164) | **DDS_NOT_NEW_VIEW_STATE** (p. 164).*

7.27.1 Detailed Description

DDS_ViewStateKind (p. 163) and associated elements

7.27.2 Typedef Documentation

7.27.2.1 DDS_ViewStateMask

```
typedef DDS_UnsignedLong DDS_ViewStateMask
```

A bit-mask (list) of view states, i.e. **DDS_ViewStateKind** (p. 163).

7.27.3 Enumeration Type Documentation

7.27.3.1 DDS_ViewStateKind

```
enum DDS_ViewStateKind
```

Indicates whether or not an instance is new.

For each instance (identified by the key), the middleware internally maintains a view state relative to each **DDSDataReader** (p. 1556). The view state can be either:

- **DDS_NEW_VIEW_STATE** (p. 164) indicates that either this is the first time that the **DDSDataReader** (p. 1556) has ever accessed samples of that instance, or else that the **DDSDataReader** (p. 1556) has accessed previous samples of the instance, but the instance has since been reborn (i.e. become not-alive and then alive again). These two cases are distinguished by examining the **DDS_SampleInfo::disposed_generation_count** (p. 1349) and the **DDS_SampleInfo::no_writers_generation_count** (p. 1349).

- **DDS_NOT_NEW_VIEW_STATE** (p. 164) indicates that the **DDSDataReader** (p. 1556) has already accessed samples of the same instance and that the instance has not been reborn since.

The view_state available in the **DDS_SampleInfo** (p. 1344) is a snapshot of the view state of the instance relative to the **DDSDataReader** (p. 1556) used to access the samples at the time the collection was obtained (i.e. at the time read or take was called). The view_state is therefore the same for all samples in the returned collection that refer to the same instance.

Once an instance has been detected as not having any "live" writers and all the samples associated with the instance are "taken" from the **DDSDataReader** (p. 1556), the middleware can reclaim all local resources regarding the instance. Future samples will be treated as "never seen."

Enumerator

DDS_NEW_VIEW_STATE	New instance. This latest generation of the instance has not previously been accessed.
DDS_NOT_NEW_VIEW_STATE	Not a new instance. This latest generation of the instance has previously been accessed.

7.27.4 Variable Documentation

7.27.4.1 DDS_ANY_VIEW_STATE

```
const DDS_ViewStateMask DDS_ANY_VIEW_STATE [extern]
```

Any view state **DDS_NEW_VIEW_STATE** (p. 164) | **DDS_NOT_NEW_VIEW_STATE** (p. 164).

Examples

HelloWorld_subscriber.cxx.

7.28 Instance States

DDS_InstanceStateKind (p. 165) and associated elements

Typedefs

- typedef **DDS_UnsignedLong DDS_InstanceStateMask**
  A bit-mask (list) of instance states, i.e. ***DDS_InstanceStateKind*** *(p. 165).*

Enumerations

- enum **DDS_InstanceStateKind** {
  DDS_ALIVE_INSTANCE_STATE = 0x0001 << 0 ,
  DDS_NOT_ALIVE_DISPOSED_INSTANCE_STATE = 0x0001 << 1 ,
  DDS_NOT_ALIVE_NO_WRITERS_INSTANCE_STATE = 0x0001 << 2 }
  Indicates if the samples are from a live ***DDSDataWriter*** *(p. 1589) or not.*

Variables

- const **DDS_InstanceStateMask DDS_ANY_INSTANCE_STATE**
  Any instance state ALIVE_INSTANCE_STATE | NOT_ALIVE_DISPOSED_INSTANCE_STATE | NOT_ALIVE_NO_↩ WRITERS_INSTANCE_STATE.
- const **DDS_InstanceStateMask DDS_NOT_ALIVE_INSTANCE_STATE**
  Not alive instance state NOT_ALIVE_DISPOSED_INSTANCE_STATE | NOT_ALIVE_NO_WRITERS_INSTANCE_STATE.

7.28.1 Detailed Description

DDS_InstanceStateKind (p. 165) and associated elements

7.28.2 Typedef Documentation

7.28.2.1 DDS_InstanceStateMask

`typedef DDS_UnsignedLong DDS_InstanceStateMask`

A bit-mask (list) of instance states, i.e. **DDS_InstanceStateKind** (p. 165).

7.28.3 Enumeration Type Documentation

7.28.3.1 DDS_InstanceStateKind

`enum DDS_InstanceStateKind`

Indicates if the samples are from a live **DDSDataWriter** (p. 1589) or not.

For each instance, the middleware internally maintains an instance state. The instance state can be:

- **DDS_ALIVE_INSTANCE_STATE** (p. 166) indicates that (a) samples have been received for the instance, (b) there are live **DDSDataWriter** (p. 1589) entities writing the instance, and (c) the instance has not been explicitly disposed (or else more samples have been received after it was disposed).
- **DDS_NOT_ALIVE_DISPOSED_INSTANCE_STATE** (p. 166) indicates the instance was explicitly disposed by a **DDSDataWriter** (p. 1589) by means of the dispose operation.
- **DDS_NOT_ALIVE_NO_WRITERS_INSTANCE_STATE** (p. 166) indicates the instance has been declared as not-alive by the **DDSDataReader** (p. 1556) because it detected that there are no live **DDSDataWriter** (p. 1589) entities writing that instance.

The precise behavior events that cause the instance state to change depends on the setting of the OWNERSHIP QoS:

- If **OWNERSHIP** (p. 612) is set to **DDS_EXCLUSIVE_OWNERSHIP_QOS** (p. 613), then the instance state becomes **DDS_NOT_ALIVE_DISPOSED_INSTANCE_STATE** (p. 166) only if the **DDSDataWriter** (p. 1589) that "owns" the instance explicitly disposes it. The instance state becomes **DDS_ALIVE_INSTANCE_STATE** (p. 166) again only if the **DDSDataWriter** (p. 1589) that owns the instance writes it.
- If **OWNERSHIP** (p. 612) is set to **DDS_SHARED_OWNERSHIP_QOS** (p. 613), then the instance state becomes **DDS_NOT_ALIVE_DISPOSED_INSTANCE_STATE** (p. 166) if any **DDSDataWriter** (p. 1589) explicitly disposes the instance. The instance state becomes **DDS_ALIVE_INSTANCE_STATE** (p. 166) as soon as any **DDSData↩ Writer** (p. 1589) writes the instance again.

The instance state available in the **DDS_SampleInfo** (p. 1344) is a snapshot of the instance state of the instance at the time the collection was obtained (i.e. at the time read or take was called). The instance state is therefore the same for all samples in the returned collection that refer to the same instance.

Enumerator

DDS_ALIVE_INSTANCE_STATE	Instance is currently in existence.
DDS_NOT_ALIVE_DISPOSED_INSTANCE_STATE	Not alive disposed instance. The instance has been disposed by a DataWriter.
DDS_NOT_ALIVE_NO_WRITERS_INSTANCE_STATE	Not alive no writers for instance. None of the **DDSDataWriter** (p. 1589) objects that are currently alive (according to the **LIVELINESS** (p. 607)) are writing the instance.

7.28.4 Variable Documentation

7.28.4.1 DDS_ANY_INSTANCE_STATE

```
const DDS_InstanceStateMask DDS_ANY_INSTANCE_STATE [extern]
```

Any instance state ALIVE_INSTANCE_STATE | NOT_ALIVE_DISPOSED_INSTANCE_STATE | NOT_ALIVE_NO_↩ WRITERS_INSTANCE_STATE.

Examples

HelloWorld_subscriber.cxx.

7.28.4.2 DDS_NOT_ALIVE_INSTANCE_STATE

```
const DDS_InstanceStateMask DDS_NOT_ALIVE_INSTANCE_STATE [extern]
```

Not alive instance state NOT_ALIVE_DISPOSED_INSTANCE_STATE | NOT_ALIVE_NO_WRITERS_INSTANCE_↩ STATE.

7.29 Stream Kinds

DDS_StreamKind (p. 167) and associated elements

Typedefs

- typedef **DDS_UnsignedLong DDS_StreamKindMask**

  A bit-mask (list) of stream kinds, i.e. ***DDS_StreamKind*** *(p. 167).*

Enumerations

- enum **DDS_StreamKind** {
  DDS_LIVE_STREAM = 0x0001 << 0 ,
  DDS_TOPIC_QUERY_STREAM = 0x0001 << 1 }
  Indicates if the samples are TopicQuery samples or not.

7.29.1 Detailed Description

DDS_StreamKind (p. 167) and associated elements

7.29.2 Typedef Documentation

7.29.2.1 DDS_StreamKindMask

`typedef` **`DDS_UnsignedLong DDS_StreamKindMask`**

A bit-mask (list) of stream kinds, i.e. **DDS_StreamKind** (p. 167).

7.29.3 Enumeration Type Documentation

7.29.3.1 DDS_StreamKind

`enum` **`DDS_StreamKind`**

Indicates if the samples are TopicQuery samples or not.

Enumerator

DDS_LIVE_STREAM	Sample is a live data sample.
DDS_TOPIC_QUERY_STREAM	Sample is a TopicQuery sample.

7.30 Infrastructure Module

Defines the abstract classes and the interfaces that are refined by the other modules. Contains common definitions such as return codes, status values, and QoS policies.

Modules

- **Clock Selection**

  APIs related to clock selection.
- **Time Support**

  Time and duration types and defines.
- **GUID Support**

  *<<**extension**>> (p. 432) GUID type and defines.*
- **Sequence Number Support**

  *<<**extension**>> (p. 432) Sequence number type and defines.*
- **Exception Codes**

  *<<**extension**>> (p. 432) Exception codes.*
- **Return Codes**

  Types of return codes.
- **Status Kinds**

  Kinds of communication status.
- **QoS**

  Quality of Service (QoS) policies.
- **Entity Support**

  ***DDSEntity** (p. 1731), **DDSListener** (p. 1794) and related items.*
- **Conditions and WaitSets**

  ***DDSCondition** (p. 1544) and **DDSWaitSet** (p. 1897) and related items.*
- **Cookie**

  *<<**extension**>> (p. 432) Unique identifier for a written data sample*
- **Sample Flags**

  *<<**extension**>> (p. 432) Flags for samples.*
- **WriteParams**

  *<<**extension**>> (p. 432)*
- **Heap Support in C**

  *<<**extension**>> (p. 432) Heap allocation and free routines in C*
- **Builtin Qos Profiles**

  *<<**extension**>> (p. 432) QoS libraries, profiles, and snippets that are automatically built into RTI Connext.*
- **User-managed Threads**

  User-managed thread infrastructure.
- **Octet Buffer Support**

  *<<**extension**>> (p. 432) Octet buffer creation, cloning, and deletion.*
- **Sequence Support**

  *The **FooSeq** (p. 1965) interface allows you to work with variable-length collections of homogeneous data.*
- **String Support**

  *<<**extension**>> (p. 432) String creation, cloning, assignment, and deletion.*

Classes

- struct **DDS_QosPrintAll_t**

  Special type which is used to select the to_string overloads when printing Qos objects.

Variables

- const **DDS_QosPrintAll_t DDS_QOS_PRINT_ALL**

  Sentinel value that is used to select the to_string overload which prints all of the values of a Qos object.

7.30.1 Detailed Description

Defines the abstract classes and the interfaces that are refined by the other modules. Contains common definitions such as return codes, status values, and QoS policies.

"DCPS Infrastructure package"

7.30.2 Variable Documentation

7.30.2.1 DDS_QOS_PRINT_ALL

`const` **`DDS_QosPrintAll_t`** `DDS_QOS_PRINT_ALL [extern]`

Sentinel value that is used to select the to_string overload which prints all of the values of a Qos object.

This sentinel value is used to select the to_string overload which prints the entirety of a Qos object (e.g., **DDS_DataReaderQos::to_string(char*, DDS_UnsignedLong&, const DDS_QosPrintAll_t&) const** (p. 150)).

See also

DDS_QosPrintAll_t (p. 1285)

7.31 Built-in Sequences

Defines sequences of primitive data type. .

Classes

- struct **DDS_CharSeq**

  Instantiates ***FooSeq*** *(p. 1965) < **DDS_Char** (p. 513) >*
- struct **DDS_WcharSeq**

  Instantiates ***FooSeq*** *(p. 1965) < **DDS_Wchar** (p. 514) >*
- struct **DDS_OctetSeq**

  Instantiates ***FooSeq*** *(p. 1965) < **DDS_Octet** (p. 514) >*
- struct **DDS_UInt8Seq**

  Instantiates ***FooSeq*** *(p. 1965) < **DDS_UInt8** (p. 514) >*
- struct **DDS_Int8Seq**

  Instantiates ***FooSeq*** *(p. 1965) < **DDS_Int8** (p. 514) >*
- struct **DDS_ShortSeq**

  Instantiates ***FooSeq*** *(p. 1965) < **DDS_Short** (p. 514) >*
- struct **DDS_UnsignedShortSeq**

  Instantiates ***FooSeq*** *(p. 1965) < **DDS_UnsignedShort** (p. 514) >*
- struct **DDS_LongSeq**

  Instantiates ***FooSeq*** *(p. 1965) < **DDS_Long** (p. 515) >*
- struct **DDS_UnsignedLongSeq**

  Instantiates ***FooSeq*** *(p. 1965) < **DDS_UnsignedLong** (p. 515) >*
- struct **DDS_LongLongSeq**

  Instantiates ***FooSeq*** *(p. 1965) < **DDS_LongLong** (p. 515) >*
- struct **DDS_UnsignedLongLongSeq**

  Instantiates ***FooSeq*** *(p. 1965) < **DDS_UnsignedLongLong** (p. 515) >*
- struct **DDS_FloatSeq**

  Instantiates ***FooSeq*** *(p. 1965) < **DDS_Float** (p. 515) >*
- struct **DDS_DoubleSeq**

  Instantiates ***FooSeq*** *(p. 1965) < **DDS_Double** (p. 515) >*
- struct **DDS_LongDoubleSeq**

  Instantiates ***FooSeq*** *(p. 1965) < **DDS_LongDouble** (p. 516) >*
- struct **DDS_BooleanSeq**

  Instantiates ***FooSeq*** *(p. 1965) < **DDS_Boolean** (p. 516) >*
- struct **DDS_StringSeq**

  Instantiates ***FooSeq*** *(p. 1965) < char* > with value type semantics.*
- struct **DDS_WstringSeq**

  Instantiates ***FooSeq*** *(p. 1965) < **DDS_Wchar** (p. 514)* >*

7.31.1 Detailed Description

Defines sequences of primitive data type. .

7.32 Multi-channel DataWriters

APIs related to Multi-channel DataWriters.

APIs related to Multi-channel DataWriters.

7.32.1 What is a Multi-channel DataWriter?

A Multi-channel **DDSDataWriter** (p. 1589) is a **DDSDataWriter** (p. 1589) that is configured to send data over multiple multicast addresses, according to some filtering criteria applied to the data.

To determine which multicast addresses will be used to send the data, the middleware evaluates a set of filters that are configured for the **DDSDataWriter** (p. 1589). Each filter "guards" a channel (a set of multicast addresses). Each time a multi-channel **DDSDataWriter** (p. 1589) writes data, the filters are applied. If a filter evaluates to true, the data is sent over that filter's associated channel (set of multicast addresses). We refer to this type of filter as a Channel Guard filter.

7.32.2 Configuration on the Writer Side

To configure a multi-channel **DDSDataWriter** (p. 1589), simply define a list of all its channels in the **DDS_Multi↩ ChannelQosPolicy** (p. 1214).

The **DDS_MultiChannelQosPolicy** (p. 1214) is propagated along with discovery traffic. The value of this policy is available in **DDS_PublicationBuiltinTopicData::locator_filter** (p. 1271).

7.32.3 Configuration on the Reader Side

No special changes are required in a subscribing application to get data from a multichannel **DDSDataWriter** (p. 1589). If you want the **DDSDataReader** (p. 1556) to subscribe to only a subset of the channels, use a **DDSContentFiltered↩ Topic** (p. 1551).

For more information on Multi-channel DataWriters, refer to the `User's Manual`.

7.32.4 Reliability with Multi-Channel DataWriters

7.32.4.1 Reliable Delivery

Reliable delivery is only guaranteed when the **DDS_PresentationQosPolicy::access_scope** (p. 1252) is set to **DDS↩ _INSTANCE_PRESENTATION_QOS** (p. 616) and the filters in **DDS_MultiChannelQosPolicy** (p. 1214) are keyed-only based.

If any of the guard filters are based on non-key fields, RTI Connext only guarantees reception of the most recent data from the MultiChannel DataWriter.

7.32.4.2 Reliable Protocol Considerations

Reliability is maintained on a per-channel basis. Each channel has its own reliability channel send queue. The size of that queue is limited by **DDS_ResourceLimitsQosPolicy::max_samples** (p. 1311) and/or **DDS_DataWriter↩ ResourceLimitsQosPolicy::max_batches** (p. 921).

The protocol parameters described in **DDS_DataWriterProtocolQosPolicy** (p. 892) are applied per channel, with the following exceptions:

DDS_RtpsReliableWriterProtocol_t::low_watermark (p. 1321) and **DDS_RtpsReliableWriterProtocol_t::high_↩ watermark** (p. 1322): The low watermark and high watermark control the queue levels (in number of samples) that determine when to switch between regular and fast heartbeat rates. With MultiChannel DataWriters, high_watermark and low_watermark refer to the DataWriter's queue (not the reliability channel queue). Therefore, periodic heartbeating cannot be controlled on a per-channel basis.

Important: With MultiChannel DataWriters, low_watermark and high_watermark refer to application samples even if batching is enabled. This behavior differs from the one without MultiChannel DataWriters (where low_watermark and high_watermark refer to batches).

DDS_RtpsReliableWriterProtocol_t::heartbeats_per_max_samples (p. 1327): This field defines the number of heartbeats per send queue. For MultiChannel DataWriters, the value is applied per channel. However, the send queue size that is used to calculate the a piggyback heartbeat rate is defined per DataWriter (see **DDS_ResourceLimits↩ QosPolicy::max_samples** (p. 1311))

Important: With MultiChannel DataWriters, heartbeats_per_max_samples refers to samples even if batching is enabled. This behavior differs from the one without MultiChannels DataWriters (where heartbeats_per_max_samples refers to batches).

With batching and MultiChannel DataWriters, the size of the DataWriter's send queue should be configured using **DDS↩ _ResourceLimitsQosPolicy::max_samples** (p. 1311) instead of max_batches **DDS_DataWriterResourceLimitsQosPolicy::max_batches** (p. 921) in order to take advantage of heartbeats_per_↩ max_samples.

7.33 Transports

APIs related to RTI Connext pluggable transports.

Modules

- **Installing Transport Plugins**

  Installing and configuring transports used by RTI Connext.
- **Built-in Transport Plugins**

  Transport plugins delivered with RTI Connext.
- **Creating New Transport Plugins**

  Developing new transport plugins for RTI Connext.
- **Transport Plugins Configuration**

  Transport plugins configuration with RTI Connext.
- **Transport Address**

  Transport-independent addressing scheme using IPv6 presentation strings and numerically stored in network-ordered format.

7.33.1 Detailed Description

APIs related to RTI Connext pluggable transports.

7.33.2 Overview

RTI Connext has a pluggable transports architecture. The core of RTI Connext is transport agnostic; it does not make any assumptions about the actual transports used to send and receive messages. Instead, the RTI Connext core uses an abstract "transport API" to interact with the **transport plugins** which implement that API.

A transport plugin implements the abstract transport API and performs the actual work of sending and receiving messages over a physical transport. A collection of **builtin plugins** (see **Built-in Transport Plugins** (p. 182)) is delivered with RTI Connext for commonly used transports. New transport plugins can easily be created, thus enabling RTI Connext applications to run over transports that may not even be conceived yet. This is a powerful capability and that distinguishes RTI Connext from competing middleware approaches.

RTI Connext also provides a set of APIs for installing and configuring transport plugins to be used in an application. So that RTI Connext applications work out of the box, a subset of the builtin transport plugins is preconfigured by default (see **DDS_TransportBuiltinQosPolicy** (p. 1408)). You can "turn-off" some or all of the builtin transport plugins. In addition, you can configure other transport plugins for use by the application.

7.33.3 Transport Aliases

In order to use a transport plugin instance in an RTI Connext application, it must be registered with a **DDSDomain↩ Participant** (p. 1619). When you register a transport, you specify a sequence of "alias" strings to symbolically refer to the transport plugin. The same alias strings can be used to register more than one transport plugin.

You can register multiple transport plugins with a **DDSDomainParticipant** (p. 1619). An **alias** symbolically refers to one or more transport plugins registered with the **DDSDomainParticipant** (p. 1619). Builtin transport plugin instances can be referred to using preconfigured aliases (see **TRANSPORT_BUILTIN** (p. 641)).

A transport plugin's class name is automatically used as an implicit alias. It can be used to refer to all the transport plugin instances of that class.

You can use aliases to refer to transport plugins, in order to specify:

- the transport plugins to use for **discovery** (see **DDS_DiscoveryQosPolicy::enabled_transports** (p. 961)), and for **DDSDataWriter** (p. 1589) and **DDSDataReader** (p. 1556) entities (see **DDS_TransportSelectionQosPolicy** (p. 1421)).

- the **multicast** addresses on which to receive discovery messages (see **DDS_DiscoveryQosPolicy::multicast↩ _receive_addresses** (p. 962)), and the multicast addresses and ports on which to receive user data (see **DDS↩ _DataReaderQos::multicast** (p. 864)).

- the **unicast ports** used for user data (see **DDS_TransportUnicastQosPolicy** (p. 1422)) on both **DDSDataWriter** (p. 1589) and **DDSDataReader** (p. 1556) entities.

- the transport plugins used to parse an address string in a locator (**Locator Format** (p. 663) and **NDDS_↩ DISCOVERY_PEERS** (p. 662)).

A **DDSDomainParticipant** (p. 1619) (and contained its entities) start using a transport plugin after the **DDSDomain↩ Participant** (p. 1619) is enabled (see **DDSEntity::enable** (p. 1734)). An entity will use *all* the transport plugins that match the specified transport QoS policy. All transport plugins are treated uniformly, regardless of how they were created or registered; there is no notion of some transports being more "special" that others.

7.33.4 Transport Lifecycle

A transport plugin is owned by whomever creates it. Thus, if you create and register a transport plugin with a **DDSDomainParticipant** (p. 1619), you are responsible for deleting it by calling its destructor. Note that builtin transport plugins (**TRANSPORT_BUILTIN** (p. 641)) and transport plugins that are loaded through the **PROPERTY** (p. 617) QoS policy (see **Loading Transport Plugins through Property QoS Policy of Domain Participant** (p. 178)) are automatically managed by RTI Connext.

A user-created transport plugin must not be deleted while it is still in use by a **DDSDomainParticipant** (p. 1619). This generally means that a user-created transport plugin instance can only be deleted after the **DDSDomainParticipant** (p. 1619) with which it was registered is deleted (see **DDSDomainParticipantFactory::delete_participant** (p. 1713)). Note that a transport plugin *cannot* be "unregistered" from a **DDSDomainParticipant** (p. 1619).

A transport plugin instance cannot be registered with more than one **DDSDomainParticipant** (p. 1619) at a time. This requirement is necessary to guarantee the multi-threaded safety of the transport API.

If the same physical transport resources are to be used with more than one **DDSDomainParticipant** (p. 1619) in the same address space, the transport plugin should be written in such a way so that it can be instantiated multiple times—once for each **DDSDomainParticipant** (p. 1619) in the address space. Note that it is always possible to write the transport plugin so that multiple transport plugin instances share the same underlying resources; however the burden (if any) of guaranteeing multi-threaded safety to access shared resource shifts to the transport plugin developer.

7.33.5 Transport Class Attributes

A transport plugin instance is associated with two kinds of attributes:

- the *class* attributes that are decided by the plugin writer; these are invariant across all instances of the transport plugin class, and
- the *instance* attributes that can be set on a per instance basis by the transport plugin user.

Every transport plugin must specify the following class attributes.

transport class id (see NDDS_Transport_Property_t::classid (p. 2045)) Identifies a transport plugin implementation class. It denotes a unique "class" to which the transport plugin instance belongs. The class is used to distinguish between different transport plugin implementations. Thus, a transport plugin vendor should ensure that its transport plugin implementation has a unique class.

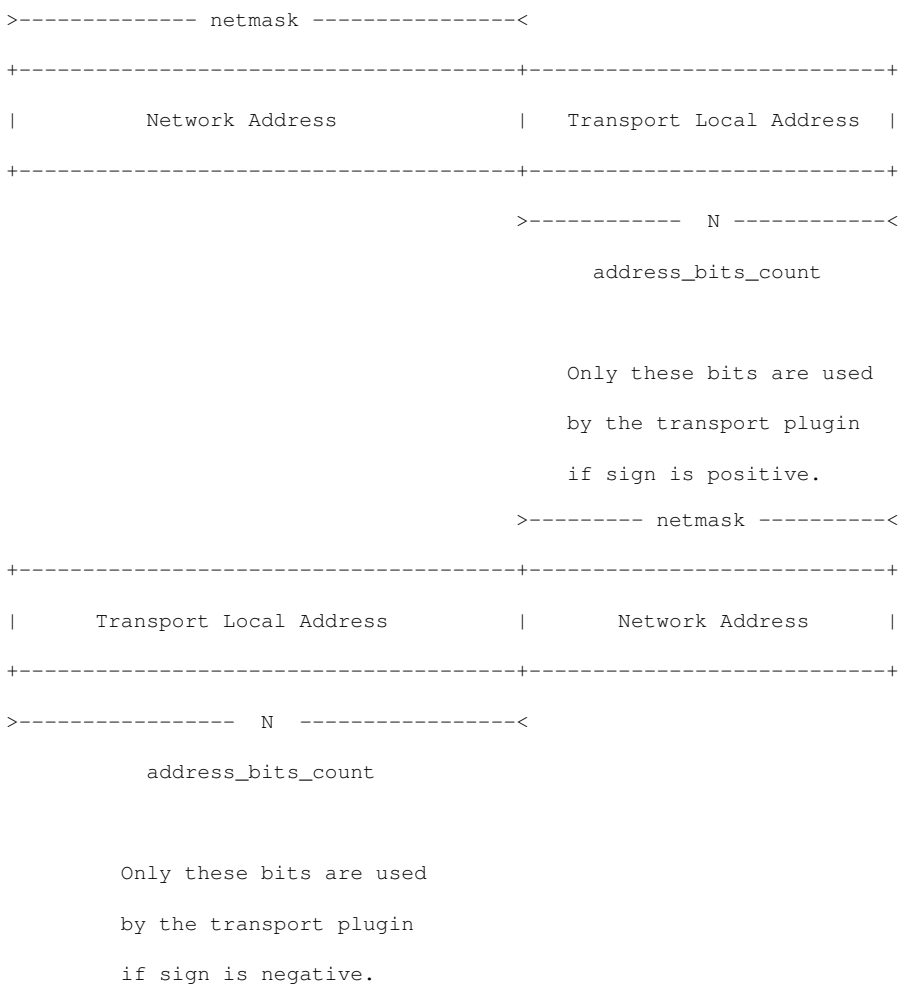
Two transport plugin instances report the same class *iff* they have compatible implementations. Transport plugin instances with mismatching classes are not allowed (by the RTI Connext Core) to communicate with one another.

Multiple implementations (possibly from different vendors) for a physical transport mechanism can co-exist in an RTI Connext application, provided they use different transport class IDs.

The class ID can also be used to distinguish between different transport protocols over the same physical transport network (e.g., UDP vs. TCP over the IP routing infrastructure).

transport significant address bit count (see NDDS_Transport_Property_t::address_bit_count (p. 2046)) RTI Connext's addressing is modeled after the IPv6 and uses 128-bit addresses (**Transport Address** (p. 448)) to route messages.

A transport plugin is expected to map the transport's internal addressing scheme to 128-bit addresses. Depending on the sign of this attribute, this mapping uses only $\mathbb{N}$ least significant bits (LSB) if positive or $\mathbb{N}$ most significant bits (MSB) if negative; these bits are specified by this attribute.



The remaining bits of an address using the 128 - abs(bit address) representation will be considered as part of the "network address" (see **Transport Network Address** (p. 176)) and thus ignored by the transport plugin's internal addressing scheme.

For *unicast* addresses, the transport plugin is expected to ignore the higher (128 - **NDDS_Transport_Property_↩ t::address_bit_count** (p. 2046)) bits. RTI Connext is free to manipulate those bits freely in the addresses passed in/out to the transport plugin APIs.

Theoretically, the significant address bits count, N is related to the size of the underlying transport network as follows:

$$address\_bits\_count >= ceil(log_2(total\_addressable\_transport\_unicast\_interfaces))$$

The equality holds when the most compact (theoretical) internal address mapping scheme is used. A practical address mapping scheme may waste some bits.

7.33.6 Transport Instance Attributes

The *per instance* attributes to configure the plugin instance are generally passed in to the plugin constructor. These are defined by the transport plugin writer, and can be used to:

- customize the behavior of an instance of a transport plugin, including the send and the receiver buffer sizes, the maximum message size, various transport level classes of service (CoS), and so on.

- specify the resource values, network interfaces to use, various transport level policies, and so on.

RTI Connext requires that every transport plugin instance must specify the **NDDS_Transport_Property_t::message_size_max** (p. 2047) and **NDDS_Transport_Property_t::gather_send_buffer_count_max** (p. 2047).

It is up to the transport plugin developer to make these available for configuration to transport plugin user.

Note that it is important that the instance attributes are "compatible" between the sending side and the receiving side of communicating applications using different instances of a transport plugin class. For example, if one side is configured to send messages larger than can be received by the other side, then communications via the plugin may fail.

7.33.7 Transport Network Address

The address bits not used by the transport plugin for its internal addressing constitute its network address bits.

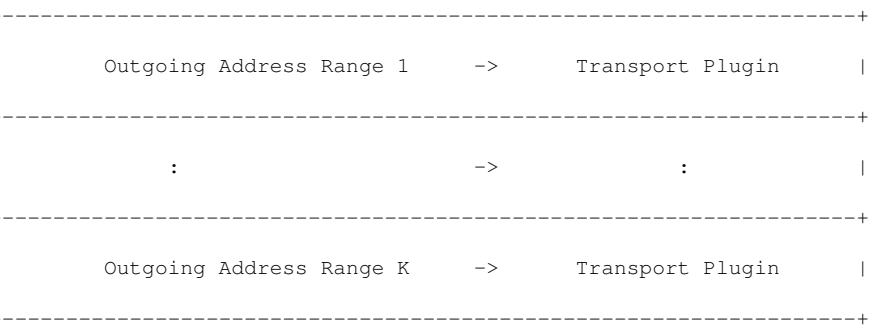
In order for RTI Connext to properly route the messages, each unicast interface in the RTI Connext *domain* must have a unique address. RTI Connext allows the user to specify the value of the network address when installing a transport plugin via the **NDDSTransportSupport::register_transport()** (p. 2104) API.

The network address for a transport plugin should be chosen such that the resulting fully qualified 128-bit address will be unique in the RTI Connext domain. Thus, if two instances of a transport plugin are registered with a **DDSDomainParticipant** (p. 1619), they will be at different network addresses in order for their unicast interfaces to have unique fully qualified 128-bit addresses. It is also possible to create multiple transports with the same network address, as it can be useful for certain use cases; note that this will require special entity configuration for most transports to avoid clashes in resource use (e.g. sockets for UDPv4 transport).

7.33.8 Transport Send Route

By default, a transport plugin is configured to send outgoing messages destined to addresses in the network address range at which the plugin was registered.

RTI Connext allows the user to configure the routing of outgoing messages via the **NDDSTransportSupport::add_send_route()** (p. 2106) API, so that a transport plugin will be used to send messages only to certain ranges of destination addresses. The method can be called multiple times for a transport plugin, with different address ranges.

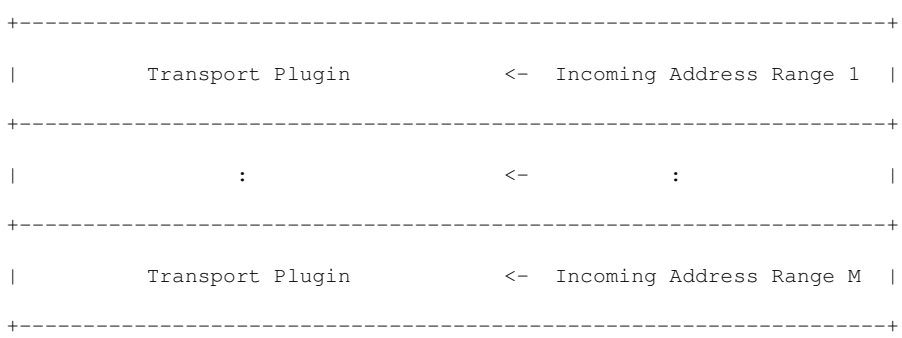


The user can set up a routing table to restrict the use of a transport plugin to send messages to selected addresses ranges.

7.33.9 Transport Receive Route

By default, a transport plugin is configured to receive incoming messages destined to addresses in the network address range at which the plugin was registered.

RTI Connext allows the user to configure the routing of incoming messages via the **NDDSTransportSupport::add_receive_route()** (p. 2107) API, so that a transport plugin will be used to receive messages only on certain ranges of addresses. The method can be called multiple times for a transport plugin, with different address ranges.



The user can set up a routing table to restrict the use of a transport plugin to receive messages from selected ranges. For example, the user may restrict a transport plugin to

- receive messages from a certain multicast address range.
- receive messages only on certain unicast interfaces (when multiple unicast interfaces are available on the transport plugin).

7.34 Installing Transport Plugins

Installing and configuring transports used by RTI Connext.

Classes

- class **NDDSTransportSupport**

  <<***interface***>> *(p. 433) The utility class used to configure RTI Connext pluggable transports.*

Typedefs

- typedef NDDS_TRANSPORT_HANDLE_TYPE_NATIVE **NDDS_Transport_Handle_t**

  *An opaque type representing the handle to a transport plugin registered with a **DDSDomainParticipant** (p. 1619).*
- typedef NDDS_Transport_Plugin *(* **NDDS_Transport_create_plugin**) (**NDDS_Transport_Address_t** *default_network_address_out, const struct **DDS_PropertyQosPolicy** *property_in)

  *Function prototype for creating plugin through **DDS_PropertyQosPolicy** (p. 1260).*

Functions

- **DDS_Boolean NDDS_Transport_Handle_is_nil** (const **NDDS_Transport_Handle_t** *self)

  Is the given transport handle the NIL transport handle?

Variables

- const **NDDS_Transport_Handle_t NDDS_TRANSPORT_HANDLE_NIL**

  The NIL transport handle.

7.34.1 Detailed Description

Installing and configuring transports used by RTI Connext.

There is more than one way to install a transport plugin for use with RTI Connext:

- If it is a builtin transport plugin, by specifying a bitmask in **DDS_TransportBuiltinQosPolicy** (p. 1408) (see **Built-in Transport Plugins** (p. 182))
- For all other non-builtin transport plugins, by dynamically loading the plugin through **PROPERTY** (p. 617) QoS policy settings of **DDSDomainParticipant** (p. 1619) (only supported on architectures that support dynamic libraries, see **Loading Transport Plugins through Property QoS Policy of Domain Participant** (p. 178))
- By explicitly creating a transport plugin and registering the plugin with a **DDSDomainParticipant** (p. 1619) through **NDDSTransportSupport::register_transport** (p. 2104) (for both builtin and non-builtin plugins)

In the first two cases, the lifecycle of the transport plugin is automatically managed by RTI Connext. In the last case, user is responsible for deleting the transport plugin after the **DDSDomainParticipant** (p. 1619) is deleted. See **Transport Lifecycle** (p. 174) for details.

7.34.2 Loading Transport Plugins through Property QoS Policy of Domain Participant

On architectures that support dynamic libraries, a non-builtin transport plugin written in C/C++ and built as a dynamic-link library (*.dll/*.so) can be loaded by RTI Connext through the **PROPERTY** (p. 617) QoS policy settings of the **DDSDomainParticipant** (p. 1619).

Dynamic libraries are supported on all architectures except INTEGRITY and certain VxWorks platforms. For VxWorks, dynamic libraries are only supported for architectures that are on Pentium/Arm CPUs AND use kernel mode.

The dynamic-link library (and all the dependent libraries) need to be in the library search path during runtime (in the **LD_LIBRARY_PATH** environment variable on Linux systems, DYLD_LIBRARY_PATH on macOS systems, or **Path** on Windows systems).

To allow dynamic loading of the transport plugin, the transport plugin must implement the RTI Connext abstract transport API and must provide a function with the signature **NDDS_Transport_create_plugin** (p. 181) that can be called by RTI Connext to create an instance of the transport plugin. The name of the dynamic library that contains the transport plugin implementation, the name of the function and properties that can be used to create the plugin, and the aliases and network address that are used to register the plugin can all be specified through the **PROPERTY** (p. 617) QoS policy of the **DDSDomainParticipant** (p. 1619).

The following table lists the property names that are used to load the transport plugins dynamically:

Table 7.73 ***Properties for dynamically loading and registering transport plugins***

Property Name	Description	Required?
dds.transport.load_plugins	Comma-separated strings indicating the prefix names of all plugins that will be loaded by RTI Connext. Up to 8 plugins may be specified. For example, "dds.transport.TCPv4.tcp1, dds.transport.TCPv4.tcp2", In the following examples, <TRANSPORT_PREFIX> is used to indicate one element of this string that is used as a prefix in the property names for all the settings that are related to the plugin. <TRANSPORT_PREFIX> must begin with "dds.transport." (such as "dds.transport.mytransport").	**YES**
<TRANSPORT_PREFIX>.library	Should be set to the name of the dynamic library (*.so for Linux systems, *.dylib for macOS systems, and *.dll for Windows systems) that contains the transport plugin implementation. This library (and all the other dependent dynamic libraries) needs to be in the library search path used by RTI Connext during run time (pointed to by the environment variable **LD_LIBRARY_PATH** on Linux systems, DYLD_LIBRARY_PATH on macOS systems, or **Path** on Windows systems).	**YES**
<TRANSPORT_PREFIX>.create_function	Should be set to the name of the function with the prototype of **NDDS_Transport_create_plugin** (p. 181) that can be called by RTI Connext to create an instance of the plugin. The resulting transport plugin will then be registered by RTI Connext through **NDDSTransportSupport::register_transport** (p. 2104)	**YES**
<TRANSPORT_PREFIX>.aliases	Used to register the transport plugin returned by **NDDS_Transport_create_plugin** (p. 181) (as specified by <**TRANSPORT_PREFIX**>**.create_function**) to the **DDSDomainParticipant** (p. 1619). Refer to **aliases_in** parameter in **NDDSTransportSupport::register_transport** (p. 2104) for details. Aliases should be specified as a comma-separated string, with each comma delimiting an alias. If it is not specified, <TRANSPORT_PREFIX> –without the leading "dds.transport" – is used as the default alias for the plugin.	NO

Property Name	Description	Required?
<TRANSPORT_PREFIX>.network_address	Used to register the transport plugin returned by **NDDS_Transport_create_plugin** (p. 181) (as specified by <**TRANSPORT_PREFIX**>**.create_function**) to the **DDSDomainParticipant** (p. 1619). Refer to **network_address_in** parameter in **NDDSTransportSupport::register_transport** (p. 2104) for details. If it is not specified, the **network_address_out** output parameter from **NDDS_Transport_create_plugin** (p. 181) is used. The default value is a zeroed out network address.	NO
<TRANSPORT_PREFIX>.<property_name>	Property that is passed into **NDDS_Transport_create_plugin** (p. 181) (as specified by <**TRANSPORT_PREFIX**>**.create_function**) for creating the transport plugin. This property name-value pair will be passed to **NDDS_Transport_create_plugin** (p. 181) after stripping out <TRANSPORT_PREFIX> from the property name. The parsing of this property and configuring the transport using this property should be handled by the implementation of each transport plugin. Multiple <TRANSPORT_PREFIX>.<property_name> can be specified. Note: "library", "create_function", "aliases" and "network_address" cannot be used as the <property_name> due to conflicts with other builtin property names.	NO

A transport plugin is dynamically created and registered to the **DDSDomainParticipant** (p. 1619) by RTI Connext when:

- the **DDSDomainParticipant** (p. 1619) is enabled,
- the first DataWriter/DataReader is created, or
- you lookup a builtin DataReader (**DDSSubscriber::lookup_datareader** (p. 1872)),

whichever happens first.

Any changes to the transport plugin related properties in the **PROPERTY** (p. 617) QoS policy after the transport plugin has been registered with the **DDSDomainParticipant** (p. 1619) will have no effect.

See also

Transport Use Cases (p. 409)

7.34.3 Typedef Documentation

7.34.3.1 NDDS_Transport_Handle_t

```
typedef NDDS_TRANSPORT_HANDLE_TYPE_NATIVE  NDDS_Transport_Handle_t
```

An opaque type representing the handle to a transport plugin registered with a **DDSDomainParticipant** (p. 1619).

A transport handle represents the association between a **DDSDomainParticipant** (p. 1619) and a transport plugin.

7.34.3.2 NDDS_Transport_create_plugin

```
typedef NDDS_Transport_Plugin *(* NDDS_Transport_create_plugin) ( NDDS_Transport_Address_t *default↩
_network_address_out, const struct  DDS_PropertyQosPolicy *property_in)
```

Function prototype for creating plugin through **DDS_PropertyQosPolicy** (p. 1260).

By specifying some predefined property names in **DDS_PropertyQosPolicy** (p. 1260), RTI Connext can call a function from a dynamic library to create a transport plugin and register the returned plugin with a **DDSDomainParticipant** (p. 1619).

This is the function prototype of the function as specified in "<TRANSPORT_PREFIX>.create_function" of **DDS_↩ PropertyQosPolicy** (p. 1260) QoS policy that will be called by RTI Connext to create the transport plugin. See **Loading Transport Plugins through Property QoS Policy of Domain Participant** (p. 178) for details.

Parameters

default_network_address_out	<<***out***>> *(p. 433)* Optional output parameter. If the network address is not specified in "<TRANSPORT_PREFIX>.network_address" in **DDS_PropertyQosPolicy** (p. 1260), this is the default network address that is used to register the returned transport plugin using **NDDSTransportSupport::register_transport** (p. 2104) . This parameter will never be null. The default value is a zeroed-out network address.
property_in	<<***in***>> *(p. 433)* **property_in** contains all the name-value pair properties that matches the format "<TRANSPORT_PREFIX>.<property_name>" in **DDS_PropertyQosPolicy** (p. 1260) that can be used to create the transport plugin. Only <property_name> is passed in - the plugin prefix will be stripped out in the property name. Note: predefined <TRANSPORT_PREFIX> properties "library", "create_function", "aliases" and "network_address" will not be passed to this function. This parameter will never be null.

Returns

Upon success, a valid non-NIL transport plugin. NIL upon failure.

7.34.4 Function Documentation

7.34.4.1 NDDS_Transport_Handle_is_nil()

```
DDS_Boolean NDDS_Transport_Handle_is_nil (
            const  NDDS_Transport_Handle_t * self )
```

Is the given transport handle the NIL transport handle?

Returns

DDS_BOOLEAN_TRUE (p. 513) if the given transport handle is equal to **NDDS_TRANSPORT_HANDLE_NIL** (p. 182) or **DDS_BOOLEAN_FALSE** (p. 513) otherwise.

7.34.5 Variable Documentation

7.34.5.1 NDDS_TRANSPORT_HANDLE_NIL

```
const  NDDS_Transport_Handle_t NDDS_TRANSPORT_HANDLE_NIL  [extern]
```

The NIL transport handle.

7.35 Built-in Transport Plugins

Transport plugins delivered with RTI Connext.

Modules

- **UDP Transport Plugin definitions**

  UDP Transport Plugin definitions.
- **Shared Memory Transport**

  *Built-in transport plug-in for inter-process communications using shared memory (**NDDS_TRANSPORT_CLASSID_SHMEM** (p. 445)) .*
- **UDPv4 Transport**

  *Transport plug-in using UDP/IPv4 (**NDDS_TRANSPORT_CLASSID_UDPv4** (p. 444)) .*
- **Real-Time WAN Transport**

  *Transport plug-in using UDP/IPv4 for WAN communications. (**NDDS_TRANSPORT_CLASSID_UDPv4_WAN** (p. 446)) .*
- **UDPv6 Transport**

  *Transport plug-in using UDP/IPv6 (**NDDS_TRANSPORT_CLASSID_UDPv6** (p. 445)) .*

7.35.1 Detailed Description

Transport plugins delivered with RTI Connext.

The **TRANSPORT_BUILTIN** (p. 641) specifies the collection of transport plugins that can be automatically configured and managed by RTI Connext as a convenience to the user.

These transport plugins can simply be turned "on" or "off" by a specifying a bitmask in **DDS_TransportBuiltinQos↩ Policy** (p. 1408), thus bypassing the steps for setting up a transport plugin. RTI Connext preconfigures the transport plugin properties, the network address, and the aliases to "factory defined" values.

If a builtin transport plugin is turned "on" in **DDS_TransportBuiltinQosPolicy** (p. 1408), the plugin is implicitly created and registered to the corresponding **DDSDomainParticipant** (p. 1619) by RTI Connext when:

- the **DDSDomainParticipant** (p. 1619) is enabled,
- the first DataWriter/DataReader is created, or
- you lookup a builtin DataReader (**DDSSubscriber::lookup_datareader** (p. 1872)),

whichever happens first.

Each builtin transport contains its own set of properties. For example, the ::**UDPv4 Transport** (p. 462) allows the application to specify whether or not multicast is supported, the maximum size of the message, and provides a mechanism for the application to filter out network interfaces.

The builtin transport plugin properties can be changed by the method **NDDSTransportSupport::set_builtin_↩ transport_property()** (p. 2109) or by using the **PROPERTY** (p. 617) QoS policy associated with the **DDSDomain↩ Participant** (p. 1619). Builtin transport plugin properties specified in **DDS_PropertyQosPolicy** (p. 1260) always overwrite the ones specified through **NDDSTransportSupport::set_builtin_transport_property()** (p. 2109). Refer to the specific builtin transport for the list of property names that can be specified through **PROPERTY** (p. 617) QoS policy.

Any changes to the builtin transport properties after the builtin transports have been registered with will have no effect.

See also

NDDSTransportSupport::set_builtin_transport_property() (p. 2109) **DDS_PropertyQosPolicy** (p. 1260)

The built-in transport plugins can also be instantiated and registered by the user, following the steps for **Registering a transport with a participant** (p. 411). This is useful when the application needs different values for the network addresses.

7.36 Creating New Transport Plugins

Developing new transport plugins for RTI Connext.

Developing new transport plugins for RTI Connext.

RTI Connext provides an abstract "C" language API for creating new transport plugins. If you are interested in creating a new transport plugin for RTI Connext, please contact your RTI representative or email `sales@rti.com`.

7.37 Queries and Filters Syntax

7.37.1 Syntax for DDS Queries and Filters

A subset of the WHERE clause in SQL is used in several parts of the specification:

- The `filter_expression` in the **DDSContentFilteredTopic** (p. 1551)
- The `query_expression` in the **DDSQueryCondition** (p. 1841)
- <<***extension***>> *(p. 432)* The `filter_expression` in the **DDS_TopicQuerySelection** (p. 1406)
- <<***extension***>> *(p. 432)* The `filter_expression` in the **DDS_ChannelSettings_t** (p. 822)

Those expressions may use a subset of SQL, extended with the possibility to use program variables in the SQL expression. The allowed SQL expressions are defined with the BNF-grammar below.

The following notational conventions are made:

- *NonTerminals* are typeset in italics.
- `'Terminals'` are quoted and typeset in a fixed width font. They are written in upper case in most cases in the BNF-grammar below, but should be case insensitive.
- **TOKENS** are typeset in bold.
- The notation (*element* // ',') represents a non-empty comma-separated list of *elements*.

7.37.2 SQL grammar in BNF

```
FilterExpression ::=  Condition
Condition   ::=  Predicate
           |    Condition  'AND'  Condition
           |    Condition  'OR'  Condition
           |    'NOT'  Condition
           |    '('  Condition  ')'
Predicate   ::=  ComparisonPredicate
           |    BetweenPredicate
ComparisonPredicate ::=  ComparisonTerm  RelOp  ComparisonTerm
ComparisonTerm      ::= FieldIdentifier
                   |  Parameter
BetweenPredicate    ::=  FieldIdentifier  'BETWEEN'  Range
                   |    FieldIdentifier  'NOT BETWEEN'  Range
FieldIdentifier      ::= FIELDNAME
                   | IDENTIFIER
RelOp     ::=  '=' |  '>' |  '>=' |  '<' |  '<=' |  '<>' |  'LIKE' |  'MATCH'
Range     ::=  Parameter  'AND'  Parameter
Parameter ::= INTEGERVALUE
         |   CHARVALUE
         |   FLOATVALUE
         |   STRING
         |   ENUMERATEDVALUE
         |   BOOLEANVALUE
         |   NULLVALUE
         |   PARAMETER
```

7.37.3 Token expression

The syntax and meaning of the tokens used in the SQL grammar is described as follows:

- **IDENTIFIER** - An identifier for a FIELDNAME, and is defined as any series of characters 'a', ..., 'z', 'A', ..., 'Z', '0', ..., '9', '_' but may not start with a digit.

Formal notation:

```
IDENTIFIER:  LETTER ( PART_LETTER)*
where  LETTER: [ "A"-"Z","_","a"-"z" ]
       PART_LETTER: [ "A"-"Z","_","a"-"z","0"-"9" ]
```

- **FIELDNAME** - A fieldname is a reference to a field in the data structure. The dot '.' is used to navigate through nested structures. The number of dots that may be used in a FIELDNAME is unlimited. The FIELDNAME can refer to fields at any depth in the data structure. The names of the field are those specified in the IDL definition of the corresponding structure, which may or may not match the fieldnames that appear on the language-specific (e.g., C/C++, Java) mapping of the structure. To reference to the *n*+1 element in an array or sequence, use the notation '[*n*]', where *n* is a natural number (zero included). FIELDNAME must resolve to a primitive IDL type; that is either boolean, octet, (unsigned) short, (unsigned) long, (unsigned) long long, float double, char, wchar, string, wstring, or enum.

Formal notation:

```
FIELDNAME:  FieldNamePart ( "."  FieldNamePart )*
where  FieldNamePart : IDENTIFIER ( "["  Index "]" )*
       Index> : (["0"-"9"])+
              | ["0x","0X"](["0"-"9", "A"-"F", "a"-"f"])+
```

Primitive IDL types referenced by FIELDNAME are treated as different types in *Predicate* according to the following table:

Predicate Data Type	IDL Type
BOOLEANVALUE	boolean
INTEGERVALUE	octet, (unsigned) short, (unsigned) long, (unsigned) long long
FLOATVALUE	float, double
CHARVALUE	char, wchar
STRING	string, wstring
ENUMERATEDVALUE	enum

- **INTEGERVALUE** - Any series of digits, optionally preceded by a plus or minus sign, representing a decimal integer value within the range of the system. 64-bit integers (int64 and uint64) must be followed by either l or L, otherwise the value is treated as a 32-bit integer. A hexadecimal number is preceded by 0x and must be a valid hexadecimal expression.

Formal notation:

```
INTEGERVALUE : (["+","-"])? (["0"-"9"])+ [("L","l")]?
             | (["+","-"])? ["0x","0X"](["0"-"9", "A"-"F", "a"-"f"])+ [("L","l")]?
```

- **CHARVALUE** - A single character enclosed between single quotes.

Formal notation:

```
CHARVALUE : "'" (~["'"])? "'"
```

- **FLOATVALUE** - Any series of digits, optionally preceded by a plus or minus sign and optionally including a floating point ('.'). A power-of-ten expression may be postfixed, which has the syntax e*n* or E*n*, where *n* is a number, optionally preceded by a plus or minus sign.

Formal notation:

```
FLOATVALUE : (["+","-"])? (["0"-"9"])* (".")? (["0"-"9"])+ ( EXPONENT)?
where  EXPONENT: ["e","E"] (["+","-"])? (["0"-"9"])+
```

- **STRING** - Any series of characters encapsulated in single quotes, except the single quote itself.

Formal notation:

```
STRING : "'" (~["'"])* "'"
```

- **ENUMERATEDVALUE** - An enumerated value is a reference to a value declared within an enumeration. Enumerated values consist of the name of the enumeration label enclosed in single quotes. The name used for the enumeration label must correspond to the label names specified in the IDL definition of the enumeration.

Formal notation:

```
ENUMERATEDVALUE : "'" ["A" - "Z", "a" - "z"] ["A" - "Z", "a" - "z", "_", "0" - "9"]* "'"
```

- **BOOLEANVALUE** - Can either be 'TRUE' or 'FALSE', case insensitive.

Formal notation (case insensitive):

```
BOOLEANVALUE : ["TRUE","FALSE"]
```

- **NULLVALUE** - Can be null, and is case insensitive.

Formal notation (case insensitive):

```
NULLVALUE : "null"
```

- **PARAMETER** - A parameter is of the form %*n*, where *n* represents a natural number (zero included) smaller than 100. It refers to the n + 1 th argument in the given context. Argument can only in primitive type value format. It cannot be a FIELDNAME.

Formal notation:

```
PARAMETER : "%" (["0"-"9"])+
```

7.37.4 String Parameters

Strings used as parameter values must contain the enclosing quotation marks (') within the parameter value, and not place the quotation marks within the expression statement. For example, the following expression is legal:

```
" symbol MATCH %0 " with parameter 0 = " 'IBM' "
```

whereas the following expression will not compile:

```
" symbol MATCH '%0' " with parameter 0 = " IBM "
```

7.37.5 Type compatability in Predicate

Only certain combination of type comparisons are valid in *Predicate*. The following table marked all the compatible pairs with 'YES':

	BOOLEANVALUE	INTEGERVALUE	FLOATVALUE	CHARVALUE	STRING	ENUMERATEDVALUE
BOOLEAN	YES					
INTEGERVALUE		YES	YES			
FLOATVALUE		YES	YES			
CHARVALUE				YES	YES	YES
STRING				YES	YES(∗1)	YES
ENUMERATEDVALUE		YES		YES(∗2)	YES(∗2)	YES(∗3)

- (∗1) See **SQL Extension: Regular Expression Matching** (p. 187)
- (∗2) Because the formal notation of the Enumeration values, they are compatible with string and char literals, but they are not compatible with string or char variables, i.e., "MyEnum='EnumValue'" would be correct, but "My↩ Enum=MyString" is not allowed.
- (∗3) Only for same type Enums.

7.37.6 SQL Extension: Regular Expression Matching

The relational operator MATCH may only be used with string fields. The right-hand operator is a string *pattern*. A string pattern specifies a template that the left-hand field value must match. The characters ,/\?∗[]-^!% have special meanings unless they are escaped by the escape character '\'.

MATCH is case-sensitive.

The pattern allows limited "wild card" matching under the following rules:

Character	Meaning
,	"," separates a list of alternate patterns. The field string is matched if it matches one or more of the patterns.
/	"/" in the pattern string matches a / in the field string. This character is used to separate a sequence of mandatory substrings.
?	"?" in the pattern string matches any single *non-special* characters in the field string.
*	"*" in the pattern string matches 0 or more *non-special* characters in field string.
[*charlist*]	Matches any one of the characters from the list of characters in *charlist*.
[*s-e*]	Matches any character any character from *s* to *e*, inclusive.
%	"%" is used to designate filter expressions parameters.
[!*charlist*] or [^*charlist*]	Matches any characters not in *charlist* (not supported).
[!*s-e*] or [^*s-e*]	Matches any characters not in the interval [s-e] (not supported).
\	Escape character for special characters.

The syntax is similar to the POSIX fnmatch syntax (1003.2-1992 section B.6). The MATCH syntax is also similar to the 'subject' strings of TIBCO Rendezvous.

Note: To use special characters as regular characters in regular expressions, you must escape them using the character '\'. For example, 'A[' is considered a malformed expression and the result is undefined.

7.37.7 Character Encoding

The default encoding for IDL strings is UTF-8. RTI Connext offers ISO 8859-1 as an alternative encoding for IDL strings.

In order to configure ISO 8859-1 as the encoding for filtering of IDL strings, you can set the DomainParticipant property **dds.domain_participant.filtering_character_encoding** to ISO-8859:

The possible values for **dds.domain_participant.filtering_character_encoding** are:

- **UTF-8** (default value)
- **ISO-8859-1**

7.37.8 Unicode Normalization

Unicode supports multiple ways to encode some characters, most notably accented characters. A composed character in Unicode can often have a number of different ways of representing the character. For example:

- Precomposed is represented by \u1e3c
- Composed = L + ^ is represented by \u004c + \u032d

The lexical comparison of the two characters above will return false. In order to do the correct comparison the characters need to be normalized, that is, reduced to the same character composition.

When the character encoding for filtering of IDL strings is UTF-8, the Unicode normalization behavior can be controlled using a DomainParticipant property called **dds.domain_participant.filtering_unicode_normalization**.

The possible values of the normalization property are:

- **OFF**: Disables normalization
- **NFD**: Canonical Decomposition
- **NFC (default value)**: Canonical Decomposition, followed by Canonical Composition
- **NFK**: Compatibility Decomposition, followed by Canonical Composition
- **NFKC_Casefold**: Casefold followed by NFKC normalization

Because normalization may affect performance, and it is enabled by default, the property allows disabling the normalization process per DomainParticipant using the value OFF. However, you should be aware that doing this may lead to unexpected behavior.

7.37.9 Examples

Assuming Topic "Location" has as an associated type a structure with fields "flight_id, x, y, z", and Topic "FlightPlan" has as fields "flight_id, source, destination". The following are examples of using these expressions.

Example of a **filter_expression** (for **DDSContentFilteredTopic** (p. 1551)) or a **query_expression** (for **DDSQuery↩ Condition** (p. 1841)):

- `"z < 1000 AND x < 23"`

Examples of a **filter_expression** using **MATCH** (for **DDSContentFilteredTopic** (p. 1551)) operator:

- `"symbol MATCH 'NASDAQ/GOOG'"`
- `"symbol MATCH 'NASDAQ/[A-M]*'"`

7.38 Logging and Version

APIs of troubleshooting utilities and APIs designed to configure the overall behavior of RTI Connext.

Modules

- **Version**

  Retrieve information for the RTI Connext product, the core library, and the C, C++ or Java libraries.
- **Logging**

  Configure how much debugging information is reported during runtime and where it is logged.

7.38.1 Detailed Description

APIs of troubleshooting utilities and APIs designed to configure the overall behavior of RTI Connext.

7.39 General Utilities and Compliance Configuration

API of general utilities used in the RTI Connext distribution.

Modules

- **Compliance Configuration**

  APIs to configure compliance with certain standard specifications.
- **Heap Monitoring**

  Monitor memory allocations done by the middleware on the native heap.
- **Network Capture**

  Save network traffic into a capture file for further analysis.
- **Other Utilities**

  Other Utilities, such as ***NDDSUtility::spin*** *(*p. *2111**).*

7.39.1 Detailed Description

API of general utilities used in the RTI Connext distribution.

7.40 Observability Framework

RTI Connext Observability Framework modules and APIs.

Modules

- **Monitoring Library 2.0**

  RTI Monitoring Library 2.0 public API.

7.40.1 Detailed Description

RTI Connext Observability Framework modules and APIs.

RTI Connext Observability Framework is a holistic solution that uses telemetry data to provide deep visibility into the current and past states of your RTI Connext applications.

7.41 Request-Reply Pattern

Support for the request-reply communication pattern.

Modules

- **Requester**

  connext::Requester *(p. 2157) and associated elements*
- **Replier**

  connext::Replier *(p. 2139),* ***connext::SimpleReplier*** *(p. 2206) and associated elements*

7.41.1 Detailed Description

Support for the request-reply communication pattern.

There are two basic entities that enable this pattern:

- **connext::Requester** (p. 2157)
- **connext::Replier** (p. 2139) (and a simpler version **connext::SimpleReplier** (p. 2206))

This functionality is built on top of RTI Connext.

A Requester publishes a request topic and subscribes to a reply topic. A Replier subscribes to the request topic and publishes the reply topic.

You can find more information about this pattern in `Request-Reply`, in the Core Libraries User's Manual.

Internal RTI Connext errors are reported as **exceptions** (p. 39).

To use request-reply in your application, include the following header file:

```
"ndds/ndds_requestreply_cpp.h"
```

See also

Request-Reply Examples (p. 419).

7.42 Requester

connext::Requester (p. 2157) and associated elements

Classes

- class **connext::Requester< TReq, TRep >**
  Allows sending requests and receiving replies.
- class **connext::RequesterParams**
  *Contains the parameters for creating a **connext::Requester*** (p. *2157*).

7.42.1 Detailed Description

connext::Requester (p. 2157) and associated elements

7.43 Replier

connext::Replier (p. 2139), **connext::SimpleReplier** (p. 2206) and associated elements

Classes

- class **connext::Replier< TReq, TRep >**
  Allows receiving requests and sending replies.
- class **connext::SimpleReplierListener< TReq, TRep >**
  *The listener called by a **SimpleReplier*** (p. *2206*).
- class **connext::ReplierListener< TReq, TRep >**
  *Called when a **connext::Replier*** (p. *2139*) *has new available requests.*
- class **connext::ReplierParams< TReq, TRep >**
  *Contains the parameters for creating a **connext::Replier*** (p. *2139*).
- class **connext::SimpleReplierParams< TReq, TRep >**
  *Contains the parameters for creating a **connext::SimpleReplier*** (p. *2206*).
- class **connext::SimpleReplier< TReq, TRep >**
  A callback-based replier.

7.43.1 Detailed Description

connext::Replier (p. 2139), **connext::SimpleReplier** (p. 2206) and associated elements

7.44 Infrastructure

Infrastructure types for Request-Reply.

Modules

- **RTI Connext Exceptions**

  RTI Connext return-code exceptions.

Classes

- class **connext::Sample**< **T** >

  A data sample and related info received from the middleware.
- class **connext::SampleRef**< **T** >

  A data sample and related information received from the middleware.
- class **connext::WriteSample**< **T** >

  A sample for writing data.
- class **connext::WriteSampleRef**< **T** >

  A reference to a data sample for writing.
- class **connext::IsValidSamplePredicate**< **T** >

  Predicate-class to determine if a sample contains valid data.
- class **connext::IsInvalidSamplePredicate**< **T** >

  Predicate-class to determine if a sample contains invalid data.
- class **connext::IsReplyRelatedPredicate**< **T** >

  Predicate-class to match replies by their related request.
- class **connext::SampleIterator**< **T, IsConst** >

  STL-compliant random-access iterator for SampleRef<T>
- class **connext::LoanedSamples**< **T** >

  Provides access to a collection of middleware-loaned samples.

Functions

- template<typename T , bool IsConst>
  ValidSampleIterator< T, IsConst > **connext::make_valid_sample_iterator** (**SampleIterator**< T, IsConst > related_iterator)

  Creates an iterator that only returns valid samples.
- template<typename T >
  LoanedSamples< T > **connext::move** (**LoanedSamples**< T > &ls) throw ()

  *Creates a new **LoanedSamples** (p. 1995) instance by moving the contents of an existing one.*

7.44.1 Detailed Description

Infrastructure types for Request-Reply.

This section describes the sample types used for reading and writing. These types encapsulate data and related information:

- **connext::Sample** (p. 2183) (used for reading)
- **connext::WriteSample** (p. 2219) (used for writing)

Sample and WriteSample are value types. They own the data and the related information and their constructor and destructor initializes and releases the data types.

There are two parallel versions that behave as reference types and hold references to existing data and information instances:

- **connext::SampleRef** (p. 2191)
- **connext::WriteSampleRef** (p. 2223)

There are different operations that provide loaned samples from the middleware. They return a **connext::LoanedSamples** (p. 1995). This container provides STL-compliant iterators.

Other convenience functions and types, useful in combination with STL algorithms are:

- **connext::make_valid_sample_iterator()** (p. 194)
- **connext::IsValidSamplePredicate** (p. 1995)
- **connext::IsReplyRelatedPredicate** (p. 1993)

7.44.2 Function Documentation

7.44.2.1 make_valid_sample_iterator()

```
template<typename T , bool IsConst>
ValidSampleIterator< T, IsConst > connext::make_valid_sample_iterator (
              SampleIterator< T, IsConst > related_iterator )
```

Creates an iterator that only returns valid samples.

This operation returns an iterator adapter that behaves as the associated **connext::SampleIterator** (p. 2191) expect that it skips samples where **SampleRef::info()** (p. 2195).valid_data is false

Parameters

related_iterator	A regular **SampleIterator** (p. 2191) (e.g. **connext::LoanedSamples::begin** (p. 2003)) that serves as the start point.

Returns

An iterator pointing to the first valid sample on or after related_iterator or the end of the associated **connext::LoanedSamples** (p. 1995) if there are no valid samples.

See also

connext::LoanedSamples::begin (p. 2003)

Taking samples by copy (p. 422)

7.44.2.2 move()

```
template<typename T >
LoanedSamples< T > connext::move (
              LoanedSamples< T > & ls ) throw ( )
```

Creates a new **LoanedSamples** (p. 1995) instance by moving the contents of an existing one.

```
The parameter object looses the ownership of the underlying samples and its state
is reset as if it was default initialized. This function must be used to move
any named LoanedSamples instance (lvalue) in and out of a function by-value.
Using this function is not necessary if the original LoanedSamples is an rvalue.
Moving is a very efficient operation and is guaranteed to not throw any exception.
```

Parameters

ls	The **LoanedSamples** (p. 1995) object that transfers its ownership of the contained samples into the returned object. After this call, `ls` is empty.

Returns

A new **LoanedSamples** (p. 1995) object, the new loan owner, with the same contents as `ls` had.

See also

connext::LoanedSamples (p. 1995)

7.45 Utilities

Utilities for the RTI Connext Messaging module.

Classes

- class **connext::MessagingLibraryVersion**

  The Connext Messaging Library version.
- class **connext::MessagingVersion**

  The Connext Messaging version.

7.45.1 Detailed Description

Utilities for the RTI Connext Messaging module.

Connext Messaging Utilities

7.46 Durability and Persistence

APIs related to RTI Connext Durability and Persistence.

APIs related to RTI Connext Durability and Persistence.

RTI Connext offers the following mechanisms for achieving durability and persistence:

- **Durable Writer History** (p. 196)
- **Durable Reader State** (p. 197)
- **Data Durability** (p. 197)

To use the first two features, you need a relational database, which is not included with RTI Connext. Supported databases are listed in the `Release Notes`.

The third feature, provided by RTI Persistence Service, can use the filesystem or a relational database to persist information.

These three features can be used separately or in combination.

7.46.1 Durable Writer History

This feature allows a **DDSDataWriter** (p. 1589) to locally persist its local history cache so that it can survive shutdowns, crashes and restarts. When an application restarts, each **DDSDataWriter** (p. 1589) that has been configured to have durable writer history automatically loads all the data in its history cache from disk and can carry on sending data as if it had never stopped executing. To the rest of the system, it will appear as if the **DDSDataWriter** (p. 1589) had been temporarily disconnected from the network and then reappeared.

See also

Configuring Durable Writer History (p. 198)

7.46.2 Durable Reader State

This feature allows a **DDSDataReader** (p. 1556) to locally persists its state and remember the sequence numbers it has already received. When an application restarts, each **DDSDataReader** (p. 1556) that has been configured to have durable reader state automatically loads its state from disk and can carry on receiving data as if it had never stopped executing. Data that had already been received by the **DDSDataReader** (p. 1556) before the restart will will not be provided to the application again.

See also

Configuring Durable Reader State (p. 200)

7.46.3 Data Durability

This feature is a full implementation of the OMG DDS Persistence Profile. The **DURABILITY** (p. 596) QoS lets an application configure a **DDSDataWriter** (p. 1589) such that the information written by the **DDSDataWriter** (p. 1589) survives beyond the lifetime of the **DDSDataWriter** (p. 1589). In this manner, a late-joining **DDSDataReader** (p. 1556) can subscribe and receive the information even after the **DDSDataWriter** (p. 1589) application is no longer executing. To use this feature, you need RTI Persistence Service – an optional product that can be purchased separately.

7.46.4 Durability and Persistence Based on Virtual GUID

Every modification to the global dataspace made by a **DDSDataWriter** (p. 1589) is identified by a pair (virtual GUID, sequence number).

- The virtual GUID (Global Unique Identifier) is a 16-byte character identifier associated with a **DDSDataWriter** (p. 1589) or **DDSDataReader** (p. 1556); it is used to uniquely identify this entity in the global data space.
- The sequence number is a 64-bit identifier that identifies changes published by a specific **DDSDataWriter** (p. 1589).

Several **DDSDataWriter** (p. 1589) entities can be configured with the same virtual GUID. If each of these **DDSDataWriter** (p. 1589) entities publishes a sample with sequence number '0', the sample will only be received once by the **DDSDataReader** (p. 1556) entities subscribing to the content published by the **DDSDataWriter** (p. 1589) entities.

RTI Connext also uses the virtual GUID (Global Unique Identifier) to associate a persisted state (state in permanent storage) to the corresponding DDS entity.

For example, the history of a **DDSDataWriter** (p. 1589) will be persisted in a database table with a name generated from the virtual GUID of the **DDSDataWriter** (p. 1589). If the **DDSDataWriter** (p. 1589) is restarted, it must have associated the same virtual GUID to restore its previous history.

Likewise, the state of a **DDSDataReader** (p. 1556) will be persisted in a database table whose name is generated from the **DDSDataReader** (p. 1556) virtual GUID

A **DDSDataWriter** (p. 1589)'s virtual GUID can be configured using **DDS_DataWriterProtocolQosPolicy::virtual_guid** (p. 893). Similarly, a **DDSDataReader** (p. 1556)'s virtual GUID can be configured using **DDS_DataReaderProtocolQosPolicy::virtual_guid** (p. 844).

The **DDS_PublicationBuiltinTopicData** (p. 1264) and **DDS_SubscriptionBuiltinTopicData** (p. 1370) structures include the virtual GUID associated with the discovered publication or subscription.

Refer to the `User's Manual` for additional use cases.

See also

DDS_DataWriterProtocolQosPolicy::virtual_guid (p. 893) **DDS_DataReaderProtocolQosPolicy::virtual_guid** (p. 844).

7.46.5 Configuring Durable Writer History

To configure a **DDSDataWriter** (p. 1589) to have durable writer history, use the **PROPERTY** (p. 617) QoS policy associated with the **DDSDataWriter** (p. 1589) or the **DDSDomainParticipant** (p. 1619).

Properties defined for the **DDSDomainParticipant** (p. 1619) will be applied to all the **DDSDataWriter** (p. 1589) objects belonging to the **DDSDomainParticipant** (p. 1619), unless the property is overwritten by the **DDSDataWriter** (p. 1589).

See also

DDS_PropertyQosPolicy (p. 1260)

The following table lists the supported durable writer history properties.

Table 7.80 ***Durable Writer History Properties***

Property	Description
dds.data_writer.history.plugin_name	Must be set to "dds.data_writer.history.odbc_plugin.builtin" to enable durable writer history in the DataWriter. This property is required.
dds.data_writer.history.odbc_plugin.builtin.dsn	The ODBC DSN (Data Source Name) associated with the database where the writer history must be persisted. This property is required.
dds.data_writer.history.odbc_plugin.builtin.driver	This property tells RTI Connext which ODBC driver to load. If the property is not specified, RTI Connext will try to use the standard ODBC driver manager library: UnixOdbc (odbc32.dll) on Linux systems; the Windows ODBC driver manager (libodbc.so) on Windows systems.
dds.data_writer.history.odbc_plugin.builtin.username	Configures the username used to connect to the database. This property is not used if it is unspecified. There is no default value.
dds.data_writer.history.odbc_plugin.builtin.password	Configures the password used to connect to the database. This property is not used if it is unspecified. There is no default value.
dds.data_writer.history.odbc_plugin.builtin.shared	If set to 1, RTI Connext creates a single connection per DSN that will be shared across DataWriters within the same Publisher. If set to 0 (the default), a **DDSDataWriter** (p. 1589) will create its own database connection. Default: 0

Property	Description
dds.data_writer.history.odbc_plugin.builtin.instance_↩ cache_max_size	These properties configure the resource limits associated with the ODBC writer history caches. To minimize the number of accesses to the database, RTI Connext uses two caches, one for samples and one for instances. The initial and maximum sizes of these caches are configured using these properties. The resource limits initial_instances, max_↩ instances, initial_samples, max_samples and max_↩ samples_per_instance in the **DDS_ResourceLimits**↩ **QosPolicy** (p. 1308) are used to configure the maximum number of samples and instances that can be stored in the relational database. Default: **DDS_Resource**↩ **LimitsQosPolicy::max_instances** (p. 1311)
dds.data_writer.history.odbc_plugin.builtin.instance_↩ cache_init_size	See description above. Default: **DDS_Resource**↩ **LimitsQosPolicy::initial_instances** (p. 1312)
dds.data_writer.history.odbc_plugin.builtin.sample_↩ cache_max_size	See description above. Default: 32 (the minimum)
dds.data_writer.history.odbc_plugin.builtin.sample_↩ cache_init_size	See description above. Default: 32
dds.data_writer.history.odbc_plugin.builtin.restore	This property indicates whether or not the persisted writer history must be restored once the **DDSDataWriter** (p. 1589) is restarted. If the value is 0, the content of the database associated with the **DDSDataWriter** (p. 1589) being restarted will be deleted. If the value is 1, the **DDSDataWriter** (p. 1589) will restore its previous state from the database content. Default: 1
dds.data_writer.history.odbc_plugin.builtin.in_memory↩ _state	This property determines how much state will be kept in memory by the ODBC writer history in order to avoid accessing the database. When in_memory_state is equal to 1, instance_↩ cache_max_size is always equal to **DDS_Resource**↩ **LimitsQosPolicy::max_instances** (p. 1311) (it cannot be changed). In addition, the ODBC writer history will keep in memory a fixed state overhead of 24 bytes per sample. In this operating mode, the ODBC writer history provides the best performance. However, the restore operation will be slower and the maximum number of samples that the writer history can manage will be limited by the available physical memory. If in_memory_state is equal to 0, all the state will be kept in the underlying database. In this operating mode, the maximum number of samples in the writer history will not be limited by the physical memory available unless the underlying database is an in-memory database (Times↩ Ten). Default: 1

7.46.6 Configuring Durable Reader State

To configure a **DDSDataReader** (p. 1556) with durable reader state, use the **PROPERTY** (p. 617) QoS policy associated with the **DDSDataReader** (p. 1556) or **DDSDomainParticipant** (p. 1619).

A property defined in the **DDSDomainParticipant** (p. 1619) will be applicable to all the **DDSDataReader** (p. 1556) belonging to the **DDSDomainParticipant** (p. 1619) unless it is overwritten by the **DDSDataReader** (p. 1556).

See also

DDS_PropertyQosPolicy (p. 1260)

The following table lists the supported durable reader state properties.

Table 7.81 ***Durable Reader State Properties***

Property	Description
dds.data_reader.state.odbc.dsn	The ODBC DSN (Data Source Name) associated with the database where the **DDSDataReader** (p. 1556) state must be persisted. This property is required.
dds.data_reader.state.filter_redundant_samples	To enable durable reader state, this property must be set to 1. Otherwise, the reader state will not be kept and/or persisted. When the reader state is not maintained, RTI Connext does not filter duplicate samples that may be coming from the same virtual writer. By default, this property is set to 1.
dds.data_reader.state.odbc.driver	This property is used to indicate which ODBC driver to load. If the property is not specified, RTI Connext will try to use the standard ODBC driver manager library↩ : UnixOdbc (odbc32.dll) on Linux systems; the Windows ODBC driver manager (libodbc.so) on Windows systems).
dds.data_reader.state.odbc.username	This property configures the username used to connect to the database. This property is not used if it is unspecified. There is no default value.
dds.data_reader.state.odbc.password	This property configures the password used to connect to the database. This property is not used if it is unspecified. There is no default value.
dds.data_reader.state.restore	This property indicates if the persisted **DDSData**↩ **Reader** (p. 1556) state must be restored or not once the **DDSDataReader** (p. 1556) is restarted. If this property is 0, the previous state will be deleted from the database. If it is 1, the **DDSDataReader** (p. 1556) will restore its previous state from the database content. Default: 1
dds.data_reader.state.checkpoint_frequency	This property controls how often the reader state is stored in the database. A value of N means to store the state once every N samples. A high value will provide better performance. However, if the reader is restarted it may receive some duplicate samples. These samples will be filtered by the middleware and they will not be propagated to the application. Default: 1

Property	Description
dds.data_reader.state.persistence_service.request_↩ depth	This property indicates the number of most recent historical samples that the persisted **DDSDataReader** (p. 1556) wants to receive when it starts up. Default: 0

7.46.7 Configuring Data Durability

RTI Connext implements **DDS_TRANSIENT_DURABILITY_QOS** (p. 599) and **DDS_PERSISTENT_DURABILITY_↩ QOS** (p. 599) durability using RTI Persistence Service, available for purchase as a separate RTI product.

For more information on RTI Persistence Service, refer to the `User's Manual`, or the RTI Persistence Service API Reference HTML documentation.

See also

DURABILITY (p. 596)

7.47 Basic QoS

QoS that are considered basic and fundamental to RTI Connext.

QoS that are considered basic and fundamental to RTI Connext.

This page lists all QoS that are considered Basic. Basic QoS are applicable to many different use cases, are widely applicable to almost any customer, and should be relatively easy to understand and set.

These are QoS policies and fields that you should be aware of even if you are not setting them today. These QoS are fundamental to DDS and to RTI Connext, and showcase RTI Connext capabilities and features.

All other Qos are applicable to a smaller subset of use cases and customers, or are finer-grained controls for a Basic QoS.

DDS_DurabilityQosPolicy (p. 1005)	**kind** (p. 1007)
	*<<**basic**>> (p. 201) <<**spec**>> (p. 209)* The kind of durability.
DDS_DurabilityQosPolicy (p. 1005)	**writer_depth** (p. 1008)
	*<<**basic**>> (p. 201) <<**extension**>> (p. 213)* Indicates the number of samples per instance that a durable **DDSDataWriter** (p. 1589) will send to a late-joining **DDSDataReader** (p. 1556).
DDS_DataRepresentationQosPolicy (p. 886)	**compression_settings** (p. 888)

	*<<**basic**>> (p. 201) <<**extension**>> (p. 213)* Structure that contains the compression settings.
DDS_DeadlineQosPolicy (p. 928)	**period** (p. 930)
	*<<**basic**>> (p. 201) <<**spec**>> (p. 209)* Duration of the deadline period.
DDS_OwnershipQosPolicy (p. 1222)	**kind** (p. 1227)
	*<<**basic**>> (p. 201) <<**spec**>> (p. 209)* The kind of ownership
DDS_OwnershipStrengthQosPolicy (p. 1227)	**value** (p. 1228)
	*<<**basic**>> (p. 201) <<**spec**>> (p. 209)* The strength value used to arbitrate among multiple writers.
DDS_LivelinessQosPolicy (p. 1180)	**kind** (p. 1182)
	*<<**basic**>> (p. 201) <<**spec**>> (p. 209)* The kind of liveliness desired.
DDS_LivelinessQosPolicy (p. 1180)	**lease_duration** (p. 1182)
	*<<**basic**>> (p. 201) <<**spec**>> (p. 209)* The duration within which a **DDSDataWriter** (p. 1589) must be asserted, or else it is assumed to be not alive.
DDS_LivelinessQosPolicy (p. 1180)	**assertions_per_lease_duration** (p. 1183)
	*<<**basic**>> (p. 201) <<**extension**>> (p. 213)* The number of assertions a **DDSDataWriter** (p. 1589) will send during a its **DDS_LivelinessQosPolicy::lease_duration** (p. 1182).
DDS_TimeBasedFilterQosPolicy (p. 1389)	**minimum_separation** (p. 1391)
	*<<**basic**>> (p. 201) <<**spec**>> (p. 209)* The minimum separation duration between subsequent samples.
DDS_PartitionQosPolicy (p. 1240)	**name** (p. 1242)
	*<<**basic**>> (p. 201) <<**spec**>> (p. 209)* A list of partition names
DDS_ReliabilityQosPolicy (p. 1295)	**kind** (p. 1298)
	*<<**basic**>> (p. 201) <<**spec**>> (p. 209)* Kind of reliability.
DDS_ReliabilityQosPolicy (p. 1295)	**max_blocking_time** (p. 1298)
	*<<**basic**>> (p. 201) <<**spec**>> (p. 209)* The maximum time a DataWriter can block on a write() call.
DDS_DestinationOrderQosPolicy (p. 931)	**kind** (p. 933)
	*<<**basic**>> (p. 201) <<**spec**>> (p. 209)* Specifies the desired kind of destination order.

DDS_HistoryQosPolicy (p. 1162)	**kind** (p. 1164)
	<<***basic***>> *(p. 201)* <<***spec***>> *(p. 209)* Specifies the kind of history to be kept.
DDS_HistoryQosPolicy (p. 1162)	**depth** (p. 1164)
	<<***basic***>> *(p. 201)* <<***spec***>> *(p. 209)* Specifies the number of samples per instance to be kept, when the `kind` is **DDS_KEEP_LAST_HISTORY_QOS** (p. 604)
DDS_ResourceLimitsQosPolicy (p. 1308)	**max_samples** (p. 1311)
	<<***basic***>> *(p. 201)* <<***spec***>> *(p. 209)* Represents the maximum samples the middleware can store for any one **DDSDataWriter** (p. 1589) (or **DDSDataReader** (p. 1556)).
DDS_ResourceLimitsQosPolicy (p. 1308)	**max_instances** (p. 1311)
	<<***basic***>> *(p. 201)* <<***spec***>> *(p. 209)* Represents the maximum number of instances a **DDSDataWriter** (p. 1589) (or **DDSDataReader** (p. 1556)) can manage.
DDS_ResourceLimitsQosPolicy (p. 1308)	**initial_samples** (p. 1312)
	<<***basic***>> *(p. 201)* <<***extension***>> *(p. 213)* Represents the initial samples the middleware will store for any one **DDSDataWriter** (p. 1589) (or **DDSDataReader** (p. 1556)).
DDS_ResourceLimitsQosPolicy (p. 1308)	**initial_instances** (p. 1312)
	<<***basic***>> *(p. 201)* <<***extension***>> *(p. 213)* Represents the initial number of instances a **DDSDataWriter** (p. 1589) (or **DDSDataReader** (p. 1556)) will manage.
DDS_EntityFactoryQosPolicy (p. 1145)	**autoenable_created_entities** (p. 1147)
	<<***basic***>> *(p. 201)* <<***spec***>> *(p. 209)* Specifies whether the entity acting as a factory automatically enables the instances it creates.
DDS_LifespanQosPolicy (p. 1175)	**duration** (p. 1176)
	<<***basic***>> *(p. 201)* <<***spec***>> *(p. 209)* Maximum duration for the data's validity.
DDS_PublishModeQosPolicy (p. 1279)	**kind** (p. 1281)
	<<***basic***>> *(p. 201)* <<***extension***>> *(p. 213)* Publishing mode
DDS_WireProtocolQosPolicy (p. 1511)	**participant_id** (p. 1514)
	<<***basic***>> *(p. 201)* <<***extension***>> *(p. 213)* A value used to distinguish among different participants belonging to the same domain on the same host.
DDS_TransportBuiltinQosPolicy (p. 1408)	**mask** (p. 1408)

	*<<**basic**>> (p. 201) <<**extension**>> (p. 213)* Specifies the built-in transports that are registered automatically when the **DDSDomainParticipant** (p. 1619) is enabled.
DDS_TransportMulticastQosPolicy (p. 1415)	**value** (p. 1416)
	*<<**basic**>> (p. 201) <<**extension**>> (p. 213)* A sequence of multicast communications settings.
DDS_DiscoveryQosPolicy (p. 960)	**initial_peers** (p. 962)
	*<<**basic**>> (p. 201) <<**extension**>> (p. 213)* Determines the initial list of peers that will be contacted by the Discovery mechanism to send announcements about the presence of this participant.
DDS_DiscoveryQosPolicy (p. 960)	**multicast_receive_addresses** (p. 962)
	*<<**basic**>> (p. 201) <<**extension**>> (p. 213)* Specifies the multicast group addresses on which discovery-related **meta-traffic** can be received by the Domain↩ Participant.
DDS_DiscoveryQosPolicy (p. 960)	**accept_unknown_peers** (p. 963)
	*<<**basic**>> (p. 201) <<**extension**>> (p. 213)* Whether to accept a new participant that is not in the initial peers list.
DDS_DiscoveryConfigQosPolicy (p. 934)	**participant_liveliness_lease_duration** (p. 937)
	*<<**basic**>> (p. 201) <<**extension**>> (p. 213)* The liveliness lease duration for the participant.
DDS_DiscoveryConfigQosPolicy (p. 934)	**participant_liveliness_assert_period** (p. 937)
	*<<**basic**>> (p. 201) <<**extension**>> (p. 213)* The period to assert liveliness for the participant.
DDS_DiscoveryConfigQosPolicy (p. 934)	**participant_announcement_period** (p. 937)
	*<<**basic**>> (p. 201) <<**extension**>> (p. 213)* The period at which a participant announces itself to potential peers when using the Simple Participant Discovery Protocol 2.0 (SPDP2).
DDS_DiscoveryConfigQosPolicy (p. 934)	**initial_participant_announcements** (p. 939)
	*<<**basic**>> (p. 201) <<**extension**>> (p. 213)* The number of initial announcements sent when a participant is first enabled.
DDS_DiscoveryConfigQosPolicy (p. 934)	**new_remote_participant_announcements** (p. 939)
	*<<**basic**>> (p. 201) <<**extension**>> (p. 213)* The number of participant announcements sent when a remote participant is newly discovered.

DDS_DiscoveryConfigQosPolicy (p. 934)	**min_initial_participant_announcement_period** (p. 940)
	*<<**basic**>> (p. 201) <<**extension**>> (p. 213)* The minimum period between initial announcements when a participant is first enabled or when a remote participant is newly discovered.
DDS_DiscoveryConfigQosPolicy (p. 934)	**max_initial_participant_announcement_period** (p. 940)
	*<<**basic**>> (p. 201) <<**extension**>> (p. 213)* The maximum period between initial announcements when a participant is first enabled or when a remote participant is newly discovered.
DDS_RtpsReliableWriterProtocol_t (p. 1320)	**heartbeat_period** (p. 1323)
	*<<**basic**>> (p. 201) <<**extension**>> (p. 213)* The period at which to send heartbeats.
DDS_RtpsReliableWriterProtocol_t (p. 1320)	**fast_heartbeat_period** (p. 1323)
	*<<**basic**>> (p. 201) <<**extension**>> (p. 213)* An alternative heartbeat period used when a reliable writer needs to flush its unacknowledged samples more quickly.
DDS_RtpsReliableWriterProtocol_t (p. 1320)	**late_joiner_heartbeat_period** (p. 1324)
	*<<**basic**>> (p. 201) <<**extension**>> (p. 213)* An alternative heartbeat period used when a reliable reader joins late and needs to be caught up on cached samples of a reliable writer more quickly than the normal heartbeat rate.
DDS_RtpsReliableWriterProtocol_t (p. 1320)	**min_send_window_size** (p. 1333)
	*<<**basic**>> (p. 201) <<**extension**>> (p. 213)* Minimum size of send window of unacknowledged samples
DDS_RtpsReliableWriterProtocol_t (p. 1320)	**max_send_window_size** (p. 1333)
	*<<**basic**>> (p. 201) <<**extension**>> (p. 213)* Maximum size of send window of unacknowledged samples
DDS_PropertyQosPolicy (p. 1260)	**value** (p. 1262)
	*<<**basic**>> (p. 201) <<**extension**>> (p. 213)* Sequence of properties.
DDS_EntityNameQosPolicy (p. 1147)	**name** (p. 1148)
	*<<**basic**>> (p. 201) <<**extension**>> (p. 213)* The name of the entity.
DDS_BatchQosPolicy (p. 810)	**enable** (p. 810)
	*<<**basic**>> (p. 201) <<**extension**>> (p. 213)* Specifies whether or not batching is enabled.

DDS_BatchQosPolicy (p. 810)	**max_data_bytes** (p. 811)
	*<<**basic**>> (p. 201) <<**extension**>> (p. 213)* The maximum cumulative length of all serialized samples in a batch.
DDS_BatchQosPolicy (p. 810)	**max_samples** (p. 811)
	*<<**basic**>> (p. 201) <<**extension**>> (p. 213)* The maximum number of samples in a batch.
DDS_ProfileQosPolicy (p. 1257)	**string_profile** (p. 1258)
	*<<**basic**>> (p. 201) <<**extension**>> (p. 213)* Sequence of strings containing a XML document to load.
DDS_ProfileQosPolicy (p. 1257)	**url_profile** (p. 1258)
	*<<**basic**>> (p. 201) <<**extension**>> (p. 213)* Sequence of **URL groups** (p. 392) containing a set of XML documents to load.
DDS_ProfileQosPolicy (p. 1257)	**ignore_user_profile** (p. 1258)
	*<<**basic**>> (p. 201) <<**extension**>> (p. 213)* Ignores the file USER_QOS_PROFILES.xml in the current working directory.
DDS_ProfileQosPolicy (p. 1257)	**ignore_environment_profile** (p. 1258)
	*<<**basic**>> (p. 201) <<**extension**>> (p. 213)* Ignores the value of the **NDDS_QOS_PROFILES** (p. 392).
DDS_ProfileQosPolicy (p. 1257)	**ignore_resource_profile** (p. 1259)
	*<<**basic**>> (p. 201) <<**extension**>> (p. 213)* Ignores the file NDDS_QOS_PROFILES.xml.
DDS_LoggingQosPolicy (p. 1188)	**verbosity** (p. 1189)
	*<<**basic**>> (p. 201) <<**extension**>> (p. 213)* The verbosities at which RTI Connext diagnostic information is logged.
DDS_LoggingQosPolicy (p. 1188)	**category** (p. 1189)
	*<<**basic**>> (p. 201) <<**extension**>> (p. 213)* Categories of logged messages.
DDS_LoggingQosPolicy (p. 1188)	**output_file** (p. 1189)
	*<<**basic**>> (p. 201) <<**extension**>> (p. 213)* Specifies the file to which log messages will be redirected to.
DDS_LoggingQosPolicy (p. 1188)	**output_file_suffix** (p. 1190)
	*<<**basic**>> (p. 201) <<**extension**>> (p. 213)* Sets the file suffix when logging to a set of files
DDS_MonitoringEventDistributionSettings (p. 1197)	**publication_period** (p. 1200)

	*<<**basic**>> (p. 201) <<**extension**>> (p. 213)* Period at which the event metric thread publishes the event metrics that have changed since the last time they were published.
DDS_MonitoringPeriodicDistributionSettings (p. 1209)	**polling_period** (p. 1210)
	*<<**basic**>> (p. 201) <<**extension**>> (p. 213)* Period at which the periodic metric thread polls and publishes the periodic metrics.
DDS_MonitoringLoggingDistributionSettings (p. 1200)	**max_historical_logs** (p. 1202)
	*<<**basic**>> (p. 201) <<**extension**>> (p. 213)* The number of log messages that RTI Monitoring Library 2.0 will keep as history.
DDS_MonitoringLoggingDistributionSettings (p. 1200)	**publication_period** (p. 1203)
	*<<**basic**>> (p. 201) <<**extension**>> (p. 213)* Period at which the logging thread publishes log messages.
DDS_MonitoringDedicatedParticipantSettings (p. 1193)	**enable** (p. 1193)
	*<<**basic**>> (p. 201) <<**extension**>> (p. 213)* Enables the use of a dedicated **DDSDomainParticipant** (p. 1619) to distribute the RTI Connext application telemetry data.
DDS_MonitoringDedicatedParticipantSettings (p. 1193)	**domain_id** (p. 1194)
	*<<**basic**>> (p. 201) <<**extension**>> (p. 213)* The domain ID used in the creation of RTI Monitoring Library 2.0 **DDSDomainParticipant** (p. 1619).
DDS_MonitoringDedicatedParticipantSettings (p. 1193)	**collector_initial_peers** (p. 1194)
	*<<**basic**>> (p. 201) <<**extension**>> (p. 213)* Determines the initial list of peers that the discovery process will contact to send announcements about the presence of the **DDS_MonitoringDistributionSettings::dedicated_participant** (p. 1196).
DDS_MonitoringLoggingForwardingSettings (p. 1204)	**middleware_forwarding_level** (p. 1204)
	*<<**basic**>> (p. 201) <<**extension**>> (p. 213)* Log messages with **NDDS_CONFIG_LOG_FACILITY_MIDDLEWARE** (p. 727) and a log level more verbose than this value will not be distributed by RTI Monitoring Library 2.0.

DDS_MonitoringLoggingForwardingSettings (p. 1204)	**security_event_forwarding_level** (p. 1204)
	*<<**basic**>> (p. 201) <<**extension**>> (p. 213)* Log messages with **NDDS_CONFIG_LOG_FACILITY_↩ SECURITY_EVENT** (p. 727) and a log level more verbose than this value will not be distributed by RTI Monitoring Library 2.0.
DDS_MonitoringLoggingForwardingSettings (p. 1204)	**service_forwarding_level** (p. 1205)
	*<<**basic**>> (p. 201) <<**extension**>> (p. 213)* Log messages with **NDDS_CONFIG_LOG_FACILITY_↩ SERVICE** (p. 727) and a log level more verbose than this value will not be distributed by RTI Monitoring Library 2.0.
DDS_MonitoringLoggingForwardingSettings (p. 1204)	**user_forwarding_level** (p. 1205)
	*<<**basic**>> (p. 201) <<**extension**>> (p. 213)* Log messages with **NDDS_CONFIG_LOG_FACILITY_↩ USER** (p. 727) and a log level more verbose than this value will not be distributed by RTI Monitoring Library 2.0.
DDS_MonitoringTelemetryData (p. 1213)	**metrics** (p. 1213)
	*<<**basic**>> (p. 201) <<**extension**>> (p. 213)* Sequence of **DDS_MonitoringMetricSelection** (p. 1206) containing the event and periodic metrics that will be collected and distributed for a given set of observable resources.
DDS_MonitoringTelemetryData (p. 1213)	**logs** (p. 1213)
	*<<**basic**>> (p. 201) <<**extension**>> (p. 213)* **DDS_MonitoringLoggingForwardingSettings** (p. 1204) containing the **NDDS_Config_Syslog↩ Verbosity** (p. 727) levels that will be forwarded for the different **NDDS_Config_LogFacility** (p. 727).
DDS_MonitoringQosPolicy (p. 1210)	**enable** (p. 1211)
	*<<**basic**>> (p. 201) <<**extension**>> (p. 213)* Enables the collection and distribution of telemetry data for an RTI Connext application using RTI Monitoring Library 2.0.
DDS_MonitoringQosPolicy (p. 1210)	**application_name** (p. 1211)
	*<<**basic**>> (p. 201) <<**extension**>> (p. 213)* The name of the resource that represents this RTI Connext application.
DDS_TopicQueryDispatchQosPolicy (p. 1404)	**publication_period** (p. 1405)
	*<<**basic**>> (p. 201) <<**extension**>> (p. 213)* Sets the periodic interval at which samples are published.

7.48 DDS Specification QoS

QoS that are defined in one of the OMG DDS Specifications.

QoS that are defined in one of the OMG DDS Specifications.

This page lists the QoS that are included in an OMG DDS specification. The standards are listed here: https://www.omg.org/spec/category/omg-data-distribution-service/.

DDS_UserDataQosPolicy (p. 1504)	**value** (p. 1505)
	<<***spec***>> *(p. 209)* a sequence of octets
DDS_GroupDataQosPolicy (p. 1160)	**value** (p. 1161)
	<<***spec***>> *(p. 209)* a sequence of octets
DDS_DurabilityQosPolicy (p. 1005)	**kind** (p. 1007)
	<<***basic***>> *(p. 201)* <<***spec***>> *(p. 209)* The kind of durability.
DDS_DurabilityServiceQosPolicy (p. 1009)	**service_cleanup_delay** (p. 1010)
	<<***spec***>> *(p. 209)* Controls when the service is able to remove all information regarding a data instances.
DDS_DurabilityServiceQosPolicy (p. 1009)	**history_kind** (p. 1010)
	<<***spec***>> *(p. 209)* The kind of history to apply in recouping durable data.
DDS_DurabilityServiceQosPolicy (p. 1009)	**history_depth** (p. 1011)
	<<***spec***>> *(p. 209)* Setting to use for the **DDS_DurabilityQosPolicy::writer_depth** (p. 1008) when recouping durable data.
DDS_DurabilityServiceQosPolicy (p. 1009)	**max_samples** (p. 1011)
	<<***spec***>> *(p. 209)* Part of resource limits QoS policy to apply when feeding a late joiner.
DDS_DurabilityServiceQosPolicy (p. 1009)	**max_instances** (p. 1011)
	<<***spec***>> *(p. 209)* Part of resource limits QoS policy to apply when feeding a late joiner.
DDS_DurabilityServiceQosPolicy (p. 1009)	**max_samples_per_instance** (p. 1012)
	<<***spec***>> *(p. 209)* Part of resource limits QoS policy to apply when feeding a late joiner.

DDS_DataRepresentationQosPolicy (p. 886)	**value** (p. 888)
	<<***spec***>> *(p. 209)* Sequence of representation identifiers.
DDS_PresentationQosPolicy (p. 1249)	**access_scope** (p. 1252)
	<<***spec***>> *(p. 209)* Determines the largest scope spanning the entities for which the order and coherency of changes can be preserved.
DDS_PresentationQosPolicy (p. 1249)	**coherent_access** (p. 1252)
	<<***spec***>> *(p. 209)* Specifies support for *coherent* access. Controls whether coherent access is supported within the scope `access_scope`.
DDS_PresentationQosPolicy (p. 1249)	**ordered_access** (p. 1252)
	<<***spec***>> *(p. 209)* Specifies support for *ordered* access to the samples received at the subscription end. Controls whether ordered access is supported within the scope `access_scope`.
DDS_DeadlineQosPolicy (p. 928)	**period** (p. 930)
	<<***basic***>> *(p. 201)* <<***spec***>> *(p. 209)* Duration of the deadline period.
DDS_LatencyBudgetQosPolicy (p. 1173)	**duration** (p. 1175)
	<<***spec***>> *(p. 209)* Duration of the maximum acceptable delay.
DDS_OwnershipQosPolicy (p. 1222)	**kind** (p. 1227)
	<<***basic***>> *(p. 201)* <<***spec***>> *(p. 209)* The kind of ownership
DDS_OwnershipStrengthQosPolicy (p. 1227)	**value** (p. 1228)
	<<***basic***>> *(p. 201)* <<***spec***>> *(p. 209)* The strength value used to arbitrate among multiple writers.
DDS_LivelinessQosPolicy (p. 1180)	**kind** (p. 1182)
	<<***basic***>> *(p. 201)* <<***spec***>> *(p. 209)* The kind of liveliness desired.
DDS_LivelinessQosPolicy (p. 1180)	**lease_duration** (p. 1182)
	<<***basic***>> *(p. 201)* <<***spec***>> *(p. 209)* The duration within which a **DDSDataWriter** (p. 1589) must be asserted, or else it is assumed to be not alive.
DDS_TimeBasedFilterQosPolicy (p. 1389)	**minimum_separation** (p. 1391)
	<<***basic***>> *(p. 201)* <<***spec***>> *(p. 209)* The minimum separation duration between subsequent samples.
DDS_PartitionQosPolicy (p. 1240)	**name** (p. 1242)

	*<<**basic**>>* (p. *201)* *<<**spec**>>* (p. *209)* A list of partition names
DDS_ReliabilityQosPolicy (p. 1295)	**kind** (p. 1298)
	*<<**basic**>>* (p. *201)* *<<**spec**>>* (p. *209)* Kind of reliability.
DDS_ReliabilityQosPolicy (p. 1295)	**max_blocking_time** (p. 1298)
	*<<**basic**>>* (p. *201)* *<<**spec**>>* (p. *209)* The maximum time a DataWriter can block on a write() call.
DDS_DestinationOrderQosPolicy (p. 931)	**kind** (p. 933)
	*<<**basic**>>* (p. *201)* *<<**spec**>>* (p. *209)* Specifies the desired kind of destination order.
DDS_DestinationOrderQosPolicy (p. 931)	**scope** (p. 933)
	*<<**spec**>>* (p. *209)* Specifies the desired scope of the source destination order.
DDS_HistoryQosPolicy (p. 1162)	**kind** (p. 1164)
	*<<**basic**>>* (p. *201)* *<<**spec**>>* (p. *209)* Specifies the kind of history to be kept.
DDS_HistoryQosPolicy (p. 1162)	**depth** (p. 1164)
	*<<**basic**>>* (p. *201)* *<<**spec**>>* (p. *209)* Specifies the number of samples per instance to be kept, when the `kind` is **DDS_KEEP_LAST_HISTORY_QOS** (p. 604)
DDS_ResourceLimitsQosPolicy (p. 1308)	**max_samples** (p. 1311)
	*<<**basic**>>* (p. *201)* *<<**spec**>>* (p. *209)* Represents the maximum samples the middleware can store for any one **DDSDataWriter** (p. 1589) (or **DDSDataReader** (p. 1556)).
DDS_ResourceLimitsQosPolicy (p. 1308)	**max_instances** (p. 1311)
	*<<**basic**>>* (p. *201)* *<<**spec**>>* (p. *209)* Represents the maximum number of instances a **DDSDataWriter** (p. 1589) (or **DDSDataReader** (p. 1556)) can manage.
DDS_ResourceLimitsQosPolicy (p. 1308)	**max_samples_per_instance** (p. 1311)
	*<<**spec**>>* (p. *209)* Represents the maximum number of samples of any one instance a **DDSDataWriter** (p. 1589) (or **DDSDataReader** (p. 1556)) can manage.
DDS_EntityFactoryQosPolicy (p. 1145)	**autoenable_created_entities** (p. 1147)
	*<<**basic**>>* (p. *201)* *<<**spec**>>* (p. *209)* Specifies whether the entity acting as a factory automatically enables the instances it creates.
DDS_TransportPriorityQosPolicy (p. 1419)	**value** (p. 1421)

	<<***spec***>> *(p. 209)* This policy is a hint to the infrastructure as to how to set the priority of the underlying transport used to send the data.
DDS_LifespanQosPolicy (p. 1175)	**duration** (p. 1176)
	<<***basic***>> *(p. 201)* <<***spec***>> *(p. 209)* Maximum duration for the data's validity.
DDS_WriterDataLifecycleQosPolicy (p. 1524)	**autodispose_unregistered_instances** (p. 1525)
	<<***spec***>> *(p. 209)* Boolean flag that controls whether or not a writer will also dispose of an instance when the **DDSDataWriter** (p. 1589) unregisters from the instance by means of the unregister operations.
DDS_ReaderDataLifecycleQosPolicy (p. 1289)	**autopurge_nowriter_samples_delay** (p. 1291)
	<<***spec***>> *(p. 209)* Minimum duration for which the **DDSDataReader** (p. 1556) will maintain samples for an instance once its `instance_state` becomes **DDS_NOT_ALIVE_NO_WRITERS_INSTANCE_STATE** (p. 166).
DDS_ReaderDataLifecycleQosPolicy (p. 1289)	**autopurge_disposed_samples_delay** (p. 1291)
	<<***spec***>> *(p. 209)* Minimum duration for which the **DDSDataReader** (p. 1556) will maintain samples for an instance once its `instance_state` becomes **DDS_NOT_ALIVE_DISPOSED_INSTANCE_↩ STATE** (p. 166).
DDS_TypeConsistencyEnforcementQosPolicy (p. 1493)	**kind** (p. 1494)
	<<***spec***>> *(p. 209)* Type consistency kind.
DDS_TypeConsistencyEnforcementQosPolicy (p. 1493)	**ignore_sequence_bounds** (p. 1495)
	<<***spec***>> *(p. 209)* Controls whether sequence bounds are taken into consideration for type assignability.
DDS_TypeConsistencyEnforcementQosPolicy (p. 1493)	**ignore_string_bounds** (p. 1495)
	<<***spec***>> *(p. 209)* Controls whether string bounds are taken into consideration for type assignability.
DDS_TypeConsistencyEnforcementQosPolicy (p. 1493)	**ignore_member_names** (p. 1495)
	<<***spec***>> *(p. 209)* Controls whether member names are taken into consideration for type assignability.
DDS_TypeConsistencyEnforcementQosPolicy (p. 1493)	**prevent_type_widening** (p. 1496)

	<<***spec***>> *(p. 209)* Controls whether type widening is allowed.
DDS_TypeConsistencyEnforcementQosPolicy (p. 1493)	**force_type_validation** (p. 1496)
	<<***spec***>> *(p. 209)* Controls whether type information must be available in order to complete matching between a **DDSDataWriter** (p. 1589) and a **DDSData↩Reader** (p. 1556).
DDS_TypeConsistencyEnforcementQosPolicy (p. 1493)	**ignore_enum_literal_names** (p. 1496)
	<<***spec***>> *(p. 209)* Controls whether enumeration constant names are taken into consideration for type assignability.
DDS_DataTags (p. 889)	**tags** (p. 889)
	<<***spec***>> *(p. 209)* Sequence of data tags.

7.49 DDS Extension QoS

QoS that are RTI extensions to the DDS standard.

QoS that are RTI extensions to the DDS standard.

This page lists all of the QoS that are RTI extensions of the DDS Standard. The extension QoS complement the standard QoS specified by the OMG DDS specifications. They are provided to improve product usability and enable access to product-specific features.

DDS_AvailabilityQosPolicy (p. 805)	**enable_required_subscriptions** (p. 808)
	<<***extension***>> *(p. 213)* Enables support for required subscriptions in a **DDSDataWriter** (p. 1589)
DDS_AvailabilityQosPolicy (p. 805)	**max_data_availability_waiting_time** (p. 808)
	<<***extension***>> *(p. 213)* Defines how much time to wait before delivering a sample to the application without having received some of the previous samples.
DDS_AvailabilityQosPolicy (p. 805)	**max_endpoint_availability_waiting_time** (p. 808)
	<<***extension***>> *(p. 213)* Defines how much time to wait to discover DataWriters providing samples for the same data source (virtual GUID).
DDS_AvailabilityQosPolicy (p. 805)	**required_matched_endpoint_groups** (p. 809)
	<<***extension***>> *(p. 213)* A sequence of endpoint groups.

DDS_DurabilityQosPolicy (p. 1005)	**direct_communication** (p. 1007)
	<<***extension***>> *(p. 213)* Indicates whether or not a TRANSIENT or PERSISTENT **DDSDataReader** (p. 1556) should receive samples directly from a TRANSIENT or PERSISTENT **DDSDataWriter** (p. 1589)
DDS_DurabilityQosPolicy (p. 1005)	**writer_depth** (p. 1008)
	<<***basic***>> *(p. 201)* <<***extension***>> *(p. 213)* Indicates the number of samples per instance that a durable **DDSDataWriter** (p. 1589) will send to a late-joining **DDSDataReader** (p. 1556).
DDS_DurabilityQosPolicy (p. 1005)	**storage_settings** (p. 1008)
	<<***extension***>> *(p. 213)* Configures durable writer history and durable reader state.
DDS_PersistentStorageSettings (p. 1242)	**enable** (p. 1244)
	<<***extension***>> *(p. 213)* Enables durable writer history in a **DDSDataWriter** (p. 1589) and durable reader state in a **DDSDataReader** (p. 1556).
DDS_PersistentStorageSettings (p. 1242)	**file_name** (p. 1244)
	<<***extension***>> *(p. 213)* The file name where the durable writer history or durable reader state will be stored.
DDS_PersistentStorageSettings (p. 1242)	**trace_file_name** (p. 1244)
	<<***extension***>> *(p. 213)* The file name where to store the SQL statements executed when loading and storing the durable writer history or durable reader state.
DDS_PersistentStorageSettings (p. 1242)	**journal_kind** (p. 1245)
	<<***extension***>> *(p. 213)* Sets the journal mode of the persistent storage.
DDS_PersistentStorageSettings (p. 1242)	**synchronization_kind** (p. 1245)
	<<***extension***>> *(p. 213)* Sets the level of synchronization with the physical disk.
DDS_PersistentStorageSettings (p. 1242)	**vacuum** (p. 1245)
	<<***extension***>> *(p. 213)* Sets the auto-vacuum status of the storage.
DDS_PersistentStorageSettings (p. 1242)	**restore** (p. 1246)
	<<***extension***>> *(p. 213)* Indicates if the persisted writer history or reader state must be restored.
DDS_PersistentStorageSettings (p. 1242)	**writer_instance_cache_allocation** (p. 1246)

	<<***extension***>> *(p. 213)* Configures the resource limits associated with the instance durable writer history cache.
DDS_PersistentStorageSettings (p. 1242)	**writer_sample_cache_allocation** (p. 1247)
	<<***extension***>> *(p. 213)* Configures the resource limits associated with the sample durable writer history cache.
DDS_PersistentStorageSettings (p. 1242)	**writer_memory_state** (p. 1248)
	<<***extension***>> *(p. 213)* Determines how much state will be kept in memory by the durable writer history in order to avoid accessing the persistent storage in disk.
DDS_PersistentStorageSettings (p. 1242)	**reader_checkpoint_frequency** (p. 1248)
	<<***extension***>> *(p. 213)* Controls how often the reader state is stored into the database.
DDS_CompressionSettings_t (p. 826)	**compression_ids** (p. 826)
	<<***extension***>> *(p. 213)* Mask that represents the compression algorithms enabled.
DDS_CompressionSettings_t (p. 826)	**writer_compression_level** (p. 827)
	<<***extension***>> *(p. 213)* The level of compression to use when compressing data.
DDS_CompressionSettings_t (p. 826)	**writer_compression_threshold** (p. 827)
	<<***extension***>> *(p. 213)* The threshold, in bytes, above which a serialized sample will be eligible to be compressed.
DDS_DataRepresentationQosPolicy (p. 886)	**compression_settings** (p. 888)
	<<***basic***>> *(p. 201)* <<***extension***>> *(p. 213)* Structure that contains the compression settings.
DDS_PresentationQosPolicy (p. 1249)	**drop_incomplete_coherent_set** (p. 1253)
	<<***extension***>> *(p. 213)* Indicates whether or not a **DDSDataReader** (p. 1556) should drop samples from an incomplete coherent set (one for which not all the samples were received). Such samples are reported as lost in the SAMPLE_LOST Status.
DDS_LivelinessQosPolicy (p. 1180)	**assertions_per_lease_duration** (p. 1183)
	<<***basic***>> *(p. 201)* <<***extension***>> *(p. 213)* The number of assertions a **DDSDataWriter** (p. 1589) will send during a its **DDS_LivelinessQosPolicy::lease_duration** (p. 1182).
DDS_ReliabilityQosPolicy (p. 1295)	**acknowledgment_kind** (p. 1298)
	<<***extension***>> *(p. 213)* Kind of reliable acknowledgment

DDS_ReliabilityQosPolicy (p. 1295)	**instance_state_consistency_kind** (p. 1299)
	<<***extension***>> *(p. 213)* Whether instance state consistency is enabled
DDS_ReliabilityQosPolicy (p. 1295)	**exact_reliability_kind_match** (p. 1299)
	<<***extension***>> *(p. 213)* Whether an exact reliability kind match is required for endpoint matching.
DDS_DestinationOrderQosPolicy (p. 931)	**source_timestamp_tolerance** (p. 933)
	<<***extension***>> *(p. 213)* Allowed tolerance between source timestamps of consecutive samples.
DDS_ResourceLimitsQosPolicy (p. 1308)	**initial_samples** (p. 1312)
	<<***basic***>> *(p. 201)* <<***extension***>> *(p. 213)* Represents the initial samples the middleware will store for any one **DDSDataWriter** (p. 1589) (or **DDSDataReader** (p. 1556)).
DDS_ResourceLimitsQosPolicy (p. 1308)	**initial_instances** (p. 1312)
	<<***basic***>> *(p. 201)* <<***extension***>> *(p. 213)* Represents the initial number of instances a **DDSDataWriter** (p. 1589) (or **DDSDataReader** (p. 1556)) will manage.
DDS_ResourceLimitsQosPolicy (p. 1308)	**instance_hash_buckets** (p. 1313)
	[DEPRECATED] (p. **384)** <<***extension***>> *(p. 213)* Number of hash buckets for instances.
DDS_TypeSupportQosPolicy (p. 1499)	**plugin_data** (p. 1500)
	[DEPRECATED] (p. **384)** <<***extension***>> *(p. 213)* Value to pass into the type plugin's de-/serialization function
DDS_TypeSupportQosPolicy (p. 1499)	**cdr_padding_kind** (p. 1500)
	[DEPRECATED] (p. **384)** <<***extension***>> *(p. 213)* Determines whether or not the padding bytes will be set to zero during CDR serialization.
DDS_PublishModeQosPolicy (p. 1279)	**kind** (p. 1281)
	<<***basic***>> *(p. 201)* <<***extension***>> *(p. 213)* Publishing mode
DDS_PublishModeQosPolicy (p. 1279)	**flow_controller_name** (p. 1281)
	<<***extension***>> *(p. 213)* Name of the associated flow controller
DDS_PublishModeQosPolicy (p. 1279)	**priority** (p. 1282)
	<<***extension***>> *(p. 213)* Publication priority
DDS_AsynchronousPublisherQosPolicy (p. 798)	**disable_asynchronous_write** (p. 800)

	[DEPRECATED] (p. **384**) <<***extension***>> *(p. 213)* Disable asynchronous publishing.
DDS_AsynchronousPublisherQosPolicy (p. 798)	**thread** (p. 800)
	<<***extension***>> *(p. 213)* Settings of the publishing thread.
DDS_AsynchronousPublisherQosPolicy (p. 798)	**disable_asynchronous_batch** (p. 800)
	[DEPRECATED] (p. **384**) <<***extension***>> *(p. 213)* Disable asynchronous batch flushing.
DDS_AsynchronousPublisherQosPolicy (p. 798)	**asynchronous_batch_thread** (p. 801)
	<<***extension***>> *(p. 213)* Settings of the batch flushing thread.
DDS_AsynchronousPublisherQosPolicy (p. 798)	**disable_topic_query_publication** (p. 801)
	[DEPRECATED] (p. **384**) <<***extension***>> *(p. 213)* Disable topic query publication.
DDS_AsynchronousPublisherQosPolicy (p. 798)	**topic_query_publication_thread** (p. 802)
	<<***extension***>> *(p. 213)* Settings of the **DDSTopic↩ Query** (p. 1895) publication thread.
DDS_WriterDataLifecycleQosPolicy (p. 1524)	**autopurge_unregistered_instances_delay** (p. 1525)
	<<***extension***>> *(p. 213)* Maximum duration for which the **DDSDataWriter** (p. 1589) will maintain information regarding an instance once it has unregistered the instance.
DDS_WriterDataLifecycleQosPolicy (p. 1524)	**autopurge_disposed_instances_delay** (p. 1526)
	<<***extension***>> *(p. 213)* Maximum duration for which the **DDSDataWriter** (p. 1589) will maintain information regarding an instance once it has disposed the instance.
DDS_ReaderDataLifecycleQosPolicy (p. 1289)	**autopurge_disposed_instances_delay** (p. 1291)
	<<***extension***>> *(p. 213)* Minimum duration for which the **DDSDataReader** (p. 1556) will maintain information about a received instance once its `instance_state` becomes **DDS_NOT_ALIVE_DISPOSED_INSTANCE↩ _STATE** (p. 166) and there are no samples for the instance in the DataReader queue.
DDS_ReaderDataLifecycleQosPolicy (p. 1289)	**autopurge_nowriter_instances_delay** (p. 1292)
	<<***extension***>> *(p. 213)* Minimum duration for which the **DDSDataReader** (p. 1556) will maintain information about a received instance once its `instance_↩ state` becomes **DDS_NOT_ALIVE_NO_WRITERS↩ _INSTANCE_STATE** (p. 166) and there are no samples for the instance in the DataReader queue.
DDS_WireProtocolQosPolicy (p. 1511)	**participant_id** (p. 1514)

	*<<**basic**>> (p. 201) <<**extension**>> (p. 213)* A value used to distinguish among different participants belonging to the same domain on the same host.
DDS_WireProtocolQosPolicy (p. 1511)	**rtps_host_id** (p. 1515)
	*<<**extension**>> (p. 213)* The RTPS Host ID of the domain participant.
DDS_WireProtocolQosPolicy (p. 1511)	**rtps_app_id** (p. 1516)
	*<<**extension**>> (p. 213)* The RTPS App ID of the domain participant.
DDS_WireProtocolQosPolicy (p. 1511)	**rtps_instance_id** (p. 1516)
	*<<**extension**>> (p. 213)* The RTPS Instance ID of the **DDSDomainParticipant** (p. 1619).
DDS_WireProtocolQosPolicy (p. 1511)	**rtps_well_known_ports** (p. 1517)
	*<<**extension**>> (p. 213)* Configures the RTPS well-known port mappings.
DDS_WireProtocolQosPolicy (p. 1511)	**rtps_reserved_port_mask** (p. 1517)
	*<<**extension**>> (p. 213)* Specifies which well-known ports to reserve when enabling the participant.
DDS_WireProtocolQosPolicy (p. 1511)	**rtps_auto_id_kind** (p. 1518)
	*<<**extension**>> (p. 213)* Kind of auto mechanism used to calculate the GUID prefix.
DDS_WireProtocolQosPolicy (p. 1511)	**compute_crc** (p. 1518)
	*<<**extension**>> (p. 213)* Adds RTPS CRC submessage to every message when this field is set to **DDS_↩ BOOLEAN_TRUE** (p. 513).
DDS_WireProtocolQosPolicy (p. 1511)	**check_crc** (p. 1518)
	*<<**extension**>> (p. 213)* Checks if the received RTPS message is valid by comparing the computed CRC with the received RTPS CRC submessage when this field is set to **DDS_BOOLEAN_TRUE** (p. 513).
DDS_TransportSelectionQosPolicy (p. 1421)	**enabled_transports** (p. 1422)
	*<<**extension**>> (p. 213)* A sequence of transport aliases that specifies the transport instances available for use by the entity.
DDS_TransportBuiltinQosPolicy (p. 1408)	**mask** (p. 1408)
	*<<**basic**>> (p. 201) <<**extension**>> (p. 213)* Specifies the built-in transports that are registered automatically when the **DDSDomainParticipant** (p. 1619) is enabled.

DDS_TransportUnicastQosPolicy (p. 1422)	**value** (p. 1424)
	<<***extension***>> *(p. 213)* A sequence of unicast communication settings.
DDS_TransportMulticastMappingQosPolicy (p. 1413)	**value** (p. 1415)
	<<***extension***>> *(p. 213)* A sequence of multicast communication mappings.
DDS_TransportMulticastQosPolicy (p. 1415)	**value** (p. 1416)
	<<***basic***>> *(p. 201)* <<***extension***>> *(p. 213)* A sequence of multicast communications settings.
DDS_TransportMulticastQosPolicy (p. 1415)	**kind** (p. 1416)
	<<***extension***>> *(p. 213)* A value that specifies a way to determine how to obtain the multicast address.
DDS_DiscoveryQosPolicy (p. 960)	**enabled_transports** (p. 961)
	<<***extension***>> *(p. 213)* The transports available for use by the Discovery mechanism.
DDS_DiscoveryQosPolicy (p. 960)	**initial_peers** (p. 962)
	<<***basic***>> *(p. 201)* <<***extension***>> *(p. 213)* Determines the initial list of peers that will be contacted by the Discovery mechanism to send announcements about the presence of this participant.
DDS_DiscoveryQosPolicy (p. 960)	**multicast_receive_addresses** (p. 962)
	<<***basic***>> *(p. 201)* <<***extension***>> *(p. 213)* Specifies the multicast group addresses on which discovery-related **meta-traffic** can be received by the Domain↩ Participant.
DDS_DiscoveryQosPolicy (p. 960)	**metatraffic_transport_priority** (p. 963)
	<<***extension***>> *(p. 213)* The transport priority to use for the Discovery meta-traffic.
DDS_DiscoveryQosPolicy (p. 960)	**accept_unknown_peers** (p. 963)
	<<***basic***>> *(p. 201)* <<***extension***>> *(p. 213)* Whether to accept a new participant that is not in the initial peers list.
DDS_DiscoveryQosPolicy (p. 960)	**enable_endpoint_discovery** (p. 964)
	[DEPRECATED] **(p. 384)** <<***extension***>> *(p. 213)* Whether to automatically enable endpoint discovery for all the remote participants.
DDS_DataReaderResourceLimitsQosPolicy (p. 867)	**max_remote_writers** (p. 870)
	<<***extension***>> *(p. 213)* The maximum number of remote writers from which a **DDSDataReader** (p. 1556) may read, including all instances.

DDS_DataReaderResourceLimitsQosPolicy (p. 867)	**max_remote_writers_per_instance** (p. 870)
	<<***extension***>> *(p. 213)* The maximum number of remote writers from which a **DDSDataReader** (p. 1556) may read a single instance.
DDS_DataReaderResourceLimitsQosPolicy (p. 867)	**max_samples_per_remote_writer** (p. 870)
	<<***extension***>> *(p. 213)* The maximum number of out-of-order samples from a given remote **DDSDataWriter** (p. 1589) that a **DDSDataReader** (p. 1556) may store when maintaining a reliable connection to the **DDSDataWriter** (p. 1589).
DDS_DataReaderResourceLimitsQosPolicy (p. 867)	**max_infos** (p. 871)
	<<***extension***>> *(p. 213)* The maximum number of info units that a **DDSDataReader** (p. 1556) can use to store **DDS_SampleInfo** (p. 1344).
DDS_DataReaderResourceLimitsQosPolicy (p. 867)	**initial_remote_writers** (p. 871)
	<<***extension***>> *(p. 213)* The initial number of remote writers from which a **DDSDataReader** (p. 1556) may read, including all instances.
DDS_DataReaderResourceLimitsQosPolicy (p. 867)	**initial_remote_writers_per_instance** (p. 872)
	<<***extension***>> *(p. 213)* The initial number of remote writers from which a **DDSDataReader** (p. 1556) may read a single instance.
DDS_DataReaderResourceLimitsQosPolicy (p. 867)	**initial_infos** (p. 872)
	<<***extension***>> *(p. 213)* The initial number of info units that a **DDSDataReader** (p. 1556) can have, which are used to store **DDS_SampleInfo** (p. 1344).
DDS_DataReaderResourceLimitsQosPolicy (p. 867)	**initial_outstanding_reads** (p. 873)
	<<***extension***>> *(p. 213)* The initial number of outstanding calls to read/take (or one of their variants) on the same **DDSDataReader** (p. 1556) for which memory has not been returned by calling **FooDataReader::return_loan** (p. 1940).
DDS_DataReaderResourceLimitsQosPolicy (p. 867)	**max_outstanding_reads** (p. 873)
	<<***extension***>> *(p. 213)* The maximum number of outstanding read/take calls (or one of their variants) on the same **DDSDataReader** (p. 1556) for which memory has not been returned by calling **FooDataReader::return_loan** (p. 1940).
DDS_DataReaderResourceLimitsQosPolicy (p. 867)	**max_samples_per_read** (p. 873)

	<<***extension***>> *(*p. *213)* The maximum number of data samples that the application can receive from the middleware in a single call to **FooDataReader::read** (p. 1919) or **FooDataReader::take** (p. 1920). If more data exists in the middleware, the application will need to issue multiple read/take calls.
DDS_DataReaderResourceLimitsQosPolicy (p. 867)	**disable_fragmentation_support** (p. 874)
	[DEPRECATED] (p. **384)** <<***extension***>> *(*p. *213)* Determines whether the **DDSDataReader** (p. 1556) can receive fragmented samples.
DDS_DataReaderResourceLimitsQosPolicy (p. 867)	**max_fragmented_samples** (p. 874)
	<<***extension***>> *(*p. *213)* The maximum number of samples for which the **DDSDataReader** (p. 1556) may store fragments at a given point in time.
DDS_DataReaderResourceLimitsQosPolicy (p. 867)	**initial_fragmented_samples** (p. 875)
	<<***extension***>> *(*p. *213)* The initial number of samples for which a **DDSDataReader** (p. 1556) may store fragments.
DDS_DataReaderResourceLimitsQosPolicy (p. 867)	**max_fragmented_samples_per_remote_writer** (p. 875)
	<<***extension***>> *(*p. *213)* The maximum number of samples per remote writer for which a **DDSDataReader** (p. 1556) may store fragments.
DDS_DataReaderResourceLimitsQosPolicy (p. 867)	**max_fragments_per_sample** (p. 876)
	<<***extension***>> *(*p. *213)* Maximum number of fragments for a single sample.
DDS_DataReaderResourceLimitsQosPolicy (p. 867)	**dynamically_allocate_fragmented_samples** (p. 876)
	<<***extension***>> *(*p. *213)* Determines whether the **DDSDataReader** (p. 1556) pre-allocates storage for storing fragmented samples.
DDS_DataReaderResourceLimitsQosPolicy (p. 867)	**max_total_instances** (p. 877)
	<<***extension***>> *(*p. *213)* Maximum number of instances for which a DataReader will keep state.
DDS_DataReaderResourceLimitsQosPolicy (p. 867)	**max_remote_virtual_writers** (p. 878)
	<<***extension***>> *(*p. *213)* The maximum number of remote virtual writers from which a **DDSDataReader** (p. 1556) may read, including all instances.
DDS_DataReaderResourceLimitsQosPolicy (p. 867)	**initial_remote_virtual_writers** (p. 878)
	<<***extension***>> *(*p. *213)* The initial number of remote virtual writers from which a **DDSDataReader** (p. 1556) may read, including all instances.

DDS_DataReaderResourceLimitsQosPolicy (p. 867)	**max_remote_virtual_writers_per_instance** (p. 879)
	<<***extension***>> *(p. 213)* The maximum number of virtual remote writers that can be associated with an instance.
DDS_DataReaderResourceLimitsQosPolicy (p. 867)	**initial_remote_virtual_writers_per_instance** (p. 879)
	<<***extension***>> *(p. 213)* The initial number of virtual remote writers per instance.
DDS_DataReaderResourceLimitsQosPolicy (p. 867)	**max_remote_writers_per_sample** (p. 880)
	<<***extension***>> *(p. 213)* The maximum number of remote writers allowed to write the same sample.
DDS_DataReaderResourceLimitsQosPolicy (p. 867)	**max_query_condition_filters** (p. 880)
	<<***extension***>> *(p. 213)* The maximum number of query condition filters a reader is allowed.
DDS_DataReaderResourceLimitsQosPolicy (p. 867)	**max_app_ack_response_length** (p. 881)
	<<***extension***>> *(p. 213)* Maximum length of application-level acknowledgment response data
DDS_DataReaderResourceLimitsQosPolicy (p. 867)	**keep_minimum_state_for_instances** (p. 881)
	<<***extension***>> *(p. 213)* Whether or not keep a minimum instance state for up to **DDS_DataReader↩ ResourceLimitsQosPolicy::max_total_instances** (p. 877).
DDS_DataReaderResourceLimitsQosPolicy (p. 867)	**initial_topic_queries** (p. 882)
	<<***extension***>> *(p. 213)* The initial number of Topic↩ Queries allocated by a **DDSDataReader** (p. 1556).
DDS_DataReaderResourceLimitsQosPolicy (p. 867)	**max_topic_queries** (p. 882)
	<<***extension***>> *(p. 213)* The maximum number of active TopicQueries that a **DDSDataReader** (p. 1556) can create.
DDS_DataReaderResourceLimitsQosPolicy (p. 867)	**shmem_ref_transfer_mode_attached_segment_↩ allocation** (p. 883)
	<<***extension***>> *(p. 213)* Allocation resource for the shared memory segments attached by the **DDSData↩ Reader** (p. 1556).
DDS_DataReaderResourceLimitsQosPolicy (p. 867)	**instance_replacement** (p. 883)
	<<***extension***>> *(p. 213)* Sets the kind of instances allowed to be replaced for each instance state (**DDS↩ _InstanceStateKind** (p. 165)) when a DataReader reaches **DDS_ResourceLimitsQosPolicy::max_↩ instances** (p. 1311).
DDS_DataReaderResourceLimitsQosPolicy (p. 867)	**autopurge_remote_not_alive_writer_delay** (p. 884)

	<<***extension***>> *(*p. *213)* Maximum duration for which the **DDSDataReader** (p. 1556) will maintain information regarding a **DDSDataWriter** (p. 1589) once the **DDSDataWriter** (p. 1589) has become not alive.
DDS_DataReaderResourceLimitsQosPolicy (p. 867)	**lazily_allocate_batch_resources** (p. 885)
	<<***extension***>> *(*p. *213)* Determines whether the **DDSDataReader** (p. 1556) pre-allocates resources for receiving batches of samples or lazily allocates them when needed.
DDS_DataWriterResourceLimitsQosPolicy (p. 917)	**initial_concurrent_blocking_threads** (p. 919)
	[DEPRECATED] (p. **384)** <<***extension***>> *(*p. *213)* The initial number of threads that are allowed to concurrently block on write call on the same **DDSDataWriter** (p. 1589).
DDS_DataWriterResourceLimitsQosPolicy (p. 917)	**max_concurrent_blocking_threads** (p. 919)
	[DEPRECATED] (p. **384)** <<***extension***>> *(*p. *213)* The maximum number of threads that are allowed to concurrently block on write call on the same **DDSDataWriter** (p. 1589).
DDS_DataWriterResourceLimitsQosPolicy (p. 917)	**max_remote_reader_filters** (p. 920)
	<<***extension***>> *(*p. *213)* The maximum number of remote DataReaders for which the **DDSDataWriter** (p. 1589) will perform content-based filtering.
DDS_DataWriterResourceLimitsQosPolicy (p. 917)	**initial_batches** (p. 920)
	<<***extension***>> *(*p. *213)* Represents the initial number of batches a **DDSDataWriter** (p. 1589) will manage.
DDS_DataWriterResourceLimitsQosPolicy (p. 917)	**max_batches** (p. 921)
	<<***extension***>> *(*p. *213)* Represents the maximum number of batches a **DDSDataWriter** (p. 1589) will manage.
DDS_DataWriterResourceLimitsQosPolicy (p. 917)	**instance_replacement** (p. 921)
	<<***extension***>> *(*p. *213)* Sets the kinds of instances allowed to be replaced when instance resource limits are reached.
DDS_DataWriterResourceLimitsQosPolicy (p. 917)	**replace_empty_instances** (p. 922)
	<<***extension***>> *(*p. *213)* Whether or not to replace empty instances during instance replacement
DDS_DataWriterResourceLimitsQosPolicy (p. 917)	**autoregister_instances** (p. 922)
	[DEPRECATED] (p. **384)** <<***extension***>> *(*p. *213)* Whether or not to automatically register new instances

DDS_DataWriterResourceLimitsQosPolicy (p. 917)	**max_virtual_writers** (p. 923)
	<<***extension***>> *(p. 213)* The maximum number of virtual writers supported by a **DDSDataWriter** (p. 1589)
DDS_DataWriterResourceLimitsQosPolicy (p. 917)	**initial_virtual_writers** (p. 923)
	<<***extension***>> *(p. 213)* The initial number of virtual writers supported by a **DDSDataWriter** (p. 1589)
DDS_DataWriterResourceLimitsQosPolicy (p. 917)	**initial_remote_readers** (p. 926)
	<<***extension***>> *(p. 213)* Controls the initial memory allocation for the matched remote readers of a **DDSDataWriter** (p. 1589).
DDS_DataWriterResourceLimitsQosPolicy (p. 917)	**max_remote_readers** (p. 923)
	<<***extension***>> *(p. 213)* The maximum number of remote readers supported by a **DDSDataWriter** (p. 1589)
DDS_DataWriterResourceLimitsQosPolicy (p. 917)	**max_app_ack_remote_readers** (p. 924)
	<<***extension***>> *(p. 213)* The maximum number of application-level acknowledging remote readers supported by a **DDSDataWriter** (p. 1589)
DDS_DataWriterResourceLimitsQosPolicy (p. 917)	**initial_active_topic_queries** (p. 924)
	<<***extension***>> *(p. 213)* Represents the initial number of active topic queries a **DDSDataWriter** (p. 1589) will manage.
DDS_DataWriterResourceLimitsQosPolicy (p. 917)	**max_active_topic_queries** (p. 924)
	<<***extension***>> *(p. 213)* Represents the maximum number of active topic queries a **DDSDataWriter** (p. 1589) will manage.
DDS_DataWriterResourceLimitsQosPolicy (p. 917)	**writer_loaned_sample_allocation** (p. 925)
	<<***extension***>> *(p. 213)* Represents the allocation settings of loaned samples managed by a **DDSData↩ Writer** (p. 1589).
DDS_DataWriterResourceLimitsQosPolicy (p. 917)	**initialize_writer_loaned_sample** (p. 925)
	<<***extension***>> *(p. 213)* Whether or not to initialize loaned samples returned by a **DDSDataWriter** (p. 1589)
DDS_DataWriterTransferModeQosPolicy (p. 927)	**shmem_ref_settings** (p. 928)
	<<***extension***>> *(p. 213)* Settings related to transferring data using shared memory references.
DDS_DataReaderProtocolQosPolicy (p. 843)	**virtual_guid** (p. 844)
	<<***extension***>> *(p. 213)* The virtual GUID (Global Unique Identifier).
DDS_DataReaderProtocolQosPolicy (p. 843)	**rtps_object_id** (p. 844)

	<<***extension***>> *(p. 213)* The RTPS Object ID.
DDS_DataReaderProtocolQosPolicy (p. 843)	**rtps_reliable_reader** (p. 846)
	<<***extension***>> *(p. 213)* RTPS protocol-related configuration settings for the RTPS reliable reader associated with a **DDSDataReader** (p. 1556). This parameter only has effect if the reader is configured with **DDS_RELIABLE_RELIABILITY_QOS** (p. 633) **DDS_↩ ReliabilityQosPolicyKind** (p. 632).
DDS_DataReaderProtocolQosPolicy (p. 843)	**expects_inline_qos** (p. 844)
	[DEPRECATED] (p. **384)** <<***extension***>> *(p. 213)* Specifies whether this DataReader expects inline QoS with every sample.
DDS_DataReaderProtocolQosPolicy (p. 843)	**disable_positive_acks** (p. 845)
	<<***extension***>> *(p. 213)* Whether the reader sends positive acknowledgements to writers.
DDS_DataReaderProtocolQosPolicy (p. 843)	**propagate_dispose_of_unregistered_instances** (p. 845)
	<<***extension***>> *(p. 213)* Indicates whether or not an instance can move to the **DDS_NOT_ALIVE_↩ DISPOSED_INSTANCE_STATE** (p. 166) state without being in the **DDS_ALIVE_INSTANCE_STATE** (p. 166) state.
DDS_DataReaderProtocolQosPolicy (p. 843)	**propagate_unregister_of_disposed_instances** (p. 846)
	<<***extension***>> *(p. 213)* Indicates whether or not an instance can move to the **DDS_NOT_ALIVE_NO↩ _WRITERS_INSTANCE_STATE** (p. 166) state directly from the **DDS_NOT_ALIVE_DISPOSED_INSTANCE↩ _STATE** (p. 166).
DDS_DataWriterProtocolQosPolicy (p. 892)	**virtual_guid** (p. 893)
	<<***extension***>> *(p. 213)* The virtual GUID (Global Unique Identifier).
DDS_DataWriterProtocolQosPolicy (p. 892)	**rtps_object_id** (p. 893)
	<<***extension***>> *(p. 213)* The RTPS Object ID.
DDS_DataWriterProtocolQosPolicy (p. 892)	**push_on_write** (p. 894)
	<<***extension***>> *(p. 213)* Whether to push sample out when write is called.
DDS_DataWriterProtocolQosPolicy (p. 892)	**disable_positive_acks** (p. 894)
	<<***extension***>> *(p. 213)* Controls whether or not the writer expects positive acknowledgements from matching readers.

DDS_DataWriterProtocolQosPolicy (p. 892)	**disable_inline_keyhash** (p. 895)
	<<***extension***>> *(p. 213)* Controls whether or not a keyhash is propagated on the wire with each sample.
DDS_DataWriterProtocolQosPolicy (p. 892)	**serialize_key_with_dispose** (p. 895)
	<<***extension***>> *(p. 213)* Controls whether or not the serialized key is propagated on the wire with dispose samples.
DDS_DataWriterProtocolQosPolicy (p. 892)	**propagate_app_ack_with_no_response** (p. 896)
	<<***extension***>> *(p. 213)* Controls whether or not a **DDSDataWriter** (p. 1589) receives **DDSDataWriter↩ Listener::on_application_acknowledgment** (p. 1617) notifications with an empty or invalid response.
DDS_DataWriterProtocolQosPolicy (p. 892)	**rtps_reliable_writer** (p. 896)
	<<***extension***>> *(p. 213)* RTPS protocol-related configuration settings for the RTPS reliable writer associated with a **DDSDataWriter** (p. 1589). This parameter only has effect if both the writer and the matching reader are configured with **DDS_RELIABLE_RELIABILITY_↩ QOS** (p. 633) **DDS_ReliabilityQosPolicyKind** (p. 632).
DDS_DataWriterProtocolQosPolicy (p. 892)	**initial_virtual_sequence_number** (p. 897)
	<<***extension***>> *(p. 213)* Determines, the initial virtual sequence number for this DataWriter.
DDS_SystemResourceLimitsQosPolicy (p. 1382)	**max_objects_per_thread** (p. 1383)
	[DEPRECATED] (p. 384) <<***extension***>> *(p. 213)* The maximum number of objects that can be stored per thread for a **DDSDomainParticipantFactory** (p. 1694).
DDS_SystemResourceLimitsQosPolicy (p. 1382)	**initial_objects_per_thread** (p. 1383)
	<<***extension***>> *(p. 213)* The number of objects per thread for a **DDSDomainParticipantFactory** (p. 1694) for which infrastructure will initially be allocated.
DDS_DomainParticipantResourceLimitsQosPolicy (p. 977)	**local_writer_allocation** (p. 981)
	<<***extension***>> *(p. 213)* Allocation settings applied to local DataWriters.
DDS_DomainParticipantResourceLimitsQosPolicy (p. 977)	**local_reader_allocation** (p. 981)
	<<***extension***>> *(p. 213)* Allocation settings applied to local DataReaders.
DDS_DomainParticipantResourceLimitsQosPolicy (p. 977)	**local_publisher_allocation** (p. 981)
	<<***extension***>> *(p. 213)* Allocation settings applied to local Publisher.

DDS_DomainParticipantResourceLimitsQosPolicy (p. 977)	**local_subscriber_allocation** (p. 981)
	<<***extension***>> *(p. 213)* Allocation settings applied to local Subscriber.
DDS_DomainParticipantResourceLimitsQosPolicy (p. 977)	**local_topic_allocation** (p. 982)
	<<***extension***>> *(p. 213)* Allocation settings applied to local Topic.
DDS_DomainParticipantResourceLimitsQosPolicy (p. 977)	**remote_writer_allocation** (p. 982)
	<<***extension***>> *(p. 213)* Allocation settings applied to remote DataWriters.
DDS_DomainParticipantResourceLimitsQosPolicy (p. 977)	**remote_reader_allocation** (p. 982)
	<<***extension***>> *(p. 213)* Allocation settings applied to remote DataReaders.
DDS_DomainParticipantResourceLimitsQosPolicy (p. 977)	**remote_participant_allocation** (p. 983)
	<<***extension***>> *(p. 213)* Allocation settings applied to remote DomainParticipants.
DDS_DomainParticipantResourceLimitsQosPolicy (p. 977)	**matching_writer_reader_pair_allocation** (p. 983)
	<<***extension***>> *(p. 213)* Allocation settings applied to matching local writer and remote/local reader pairs.
DDS_DomainParticipantResourceLimitsQosPolicy (p. 977)	**matching_reader_writer_pair_allocation** (p. 983)
	<<***extension***>> *(p. 213)* Allocation settings applied to matching local reader and remote/local writer pairs.
DDS_DomainParticipantResourceLimitsQosPolicy (p. 977)	**ignored_entity_allocation** (p. 984)
	<<***extension***>> *(p. 213)* Allocation settings applied to ignored entities.
DDS_DomainParticipantResourceLimitsQosPolicy (p. 977)	**content_filtered_topic_allocation** (p. 984)
	<<***extension***>> *(p. 213)* Allocation settings applied to content filtered topic.
DDS_DomainParticipantResourceLimitsQosPolicy (p. 977)	**content_filter_allocation** (p. 984)
	<<***extension***>> *(p. 213)* Allocation settings applied to content filter.

DDS_DomainParticipantResourceLimitsQosPolicy (p. 977)	**read_condition_allocation** (p. 985)
	<<***extension***>> *(p. 213)* Allocation settings applied to read condition pool.
DDS_DomainParticipantResourceLimitsQosPolicy (p. 977)	**query_condition_allocation** (p. 985)
	<<***extension***>> *(p. 213)* Allocation settings applied to query condition pool.
DDS_DomainParticipantResourceLimitsQosPolicy (p. 977)	**outstanding_asynchronous_sample_allocation** (p. 985)
	<<***extension***>> *(p. 213)* Allocation settings applied to the maximum number of samples (from all **DDSData↩ Writer** (p. 1589)) waiting to be asynchronously written.
DDS_DomainParticipantResourceLimitsQosPolicy (p. 977)	**flow_controller_allocation** (p. 986)
	<<***extension***>> *(p. 213)* Allocation settings applied to flow controllers.
DDS_DomainParticipantResourceLimitsQosPolicy (p. 977)	**local_writer_hash_buckets** (p. 986)
	[DEPRECATED] (p. 384) <<***extension***>> *(p. 213)* Hash_Buckets settings applied to local DataWriters.
DDS_DomainParticipantResourceLimitsQosPolicy (p. 977)	**local_reader_hash_buckets** (p. 986)
	[DEPRECATED] (p. 384) <<***extension***>> *(p. 213)* Number of hash buckets for local DataReaders.
DDS_DomainParticipantResourceLimitsQosPolicy (p. 977)	**local_publisher_hash_buckets** (p. 987)
	[DEPRECATED] (p. 384) <<***extension***>> *(p. 213)* Number of hash buckets for local Publisher.
DDS_DomainParticipantResourceLimitsQosPolicy (p. 977)	**local_subscriber_hash_buckets** (p. 987)
	[DEPRECATED] (p. 384) <<***extension***>> *(p. 213)* Number of hash buckets for local Subscriber.
DDS_DomainParticipantResourceLimitsQosPolicy (p. 977)	**local_topic_hash_buckets** (p. 987)
	[DEPRECATED] (p. 384) <<***extension***>> *(p. 213)* Number of hash buckets for local Topic.
DDS_DomainParticipantResourceLimitsQosPolicy (p. 977)	**remote_writer_hash_buckets** (p. 988)
	[DEPRECATED] (p. 384) <<***extension***>> *(p. 213)* Number of hash buckets for remote DataWriters.

DDS_DomainParticipantResourceLimitsQosPolicy (p. 977)	**remote_reader_hash_buckets** (p. 988)
	[DEPRECATED] (p. **384**) <<***extension***>> *(p. 213)* Number of hash buckets for remote DataReaders.
DDS_DomainParticipantResourceLimitsQosPolicy (p. 977)	**remote_participant_hash_buckets** (p. 988)
	[DEPRECATED] (p. **384**) <<***extension***>> *(p. 213)* Number of hash buckets for remote DomainParticipants.
DDS_DomainParticipantResourceLimitsQosPolicy (p. 977)	**matching_writer_reader_pair_hash_buckets** (p. 989)
	[DEPRECATED] (p. **384**) <<***extension***>> *(p. 213)* Number of hash buckets for matching local writer and remote/local reader pairs.
DDS_DomainParticipantResourceLimitsQosPolicy (p. 977)	**matching_reader_writer_pair_hash_buckets** (p. 989)
	[DEPRECATED] (p. **384**) <<***extension***>> *(p. 213)* Number of hash buckets for matching local reader and remote/local writer pairs.
DDS_DomainParticipantResourceLimitsQosPolicy (p. 977)	**ignored_entity_hash_buckets** (p. 989)
	[DEPRECATED] (p. **384**) <<***extension***>> *(p. 213)* Number of hash buckets for ignored entities.
DDS_DomainParticipantResourceLimitsQosPolicy (p. 977)	**content_filtered_topic_hash_buckets** (p. 990)
	[DEPRECATED] (p. **384**) <<***extension***>> *(p. 213)* Number of hash buckets for content filtered topics.
DDS_DomainParticipantResourceLimitsQosPolicy (p. 977)	**content_filter_hash_buckets** (p. 990)
	[DEPRECATED] (p. **384**) <<***extension***>> *(p. 213)* Number of hash buckets for content filters.
DDS_DomainParticipantResourceLimitsQosPolicy (p. 977)	**flow_controller_hash_buckets** (p. 990)
	[DEPRECATED] (p. **384**) <<***extension***>> *(p. 213)* Number of hash buckets for flow controllers.
DDS_DomainParticipantResourceLimitsQosPolicy (p. 977)	**max_gather_destinations** (p. 991)
	<<***extension***>> *(p. 213)* Maximum number of destinations per RTI Connext send.
DDS_DomainParticipantResourceLimitsQosPolicy (p. 977)	**participant_user_data_max_length** (p. 991)

	<<***extension***>> *(p. 213)* Maximum length of user data in **DDS_DomainParticipantQos** (p. 971) and **DDS_↩ ParticipantBuiltinTopicData** (p. 1229)
DDS_DomainParticipantResourceLimitsQosPolicy (p. 977)	**topic_data_max_length** (p. 991)
	<<***extension***>> *(p. 213)* Maximum length of topic data in **DDS_TopicQos** (p. 1397), **DDS_TopicBuiltin↩ TopicData** (p. 1391), **DDS_PublicationBuiltinTopic↩ Data** (p. 1264) and **DDS_SubscriptionBuiltinTopic↩ Data** (p. 1370)
DDS_DomainParticipantResourceLimitsQosPolicy (p. 977)	**publisher_group_data_max_length** (p. 992)
	<<***extension***>> *(p. 213)* Maximum length of group data in **DDS_PublisherQos** (p. 1276) and **DDS_↩ PublicationBuiltinTopicData** (p. 1264)
DDS_DomainParticipantResourceLimitsQosPolicy (p. 977)	**subscriber_group_data_max_length** (p. 992)
	<<***extension***>> *(p. 213)* Maximum length of group data in **DDS_SubscriberQos** (p. 1367) and **DDS_↩ SubscriptionBuiltinTopicData** (p. 1370)
DDS_DomainParticipantResourceLimitsQosPolicy (p. 977)	**writer_user_data_max_length** (p. 992)
	<<***extension***>> *(p. 213)* Maximum length of user data in **DDS_DataWriterQos** (p. 909) and **DDS_↩ PublicationBuiltinTopicData** (p. 1264)
DDS_DomainParticipantResourceLimitsQosPolicy (p. 977)	**reader_user_data_max_length** (p. 993)
	<<***extension***>> *(p. 213)* Maximum length of user data in **DDS_DataReaderQos** (p. 858) and **DDS_↩ SubscriptionBuiltinTopicData** (p. 1370)
DDS_DomainParticipantResourceLimitsQosPolicy (p. 977)	**max_partitions** (p. 993)
	<<***extension***>> *(p. 213)* Maximum number of partition name strings allowable in a **DDS_PartitionQosPolicy** (p. 1240).
DDS_DomainParticipantResourceLimitsQosPolicy (p. 977)	**max_partition_cumulative_characters** (p. 993)
	<<***extension***>> *(p. 213)* Maximum number of combined characters allowable in all partition names in a **DDS_PartitionQosPolicy** (p. 1240).
DDS_DomainParticipantResourceLimitsQosPolicy (p. 977)	**type_code_max_serialized_length** (p. 994)

	[DEPRECATED] (p. **384**) <<***extension***>> *(p. 213)* Maximum size of serialized string for type code.
DDS_DomainParticipantResourceLimitsQosPolicy (p. 977)	**type_object_max_serialized_length** (p. 994)
	<<***extension***>> *(p. 213)* The maximum length, in bytes, that the buffer to serialize a TypeObject v1 can consume.
DDS_DomainParticipantResourceLimitsQosPolicy (p. 977)	**type_object_max_deserialized_length** (p. 996)
	<<***extension***>> *(p. 213)* The maximum number of bytes that a deserialized TypeObject v1 can consume.
DDS_DomainParticipantResourceLimitsQosPolicy (p. 977)	**deserialized_type_object_dynamic_allocation_threshold** (p. 996)
	<<***extension***>> *(p. 213)* A threshold, in bytes, for dynamic memory allocation for the deserialized TypeObject v1.
DDS_DomainParticipantResourceLimitsQosPolicy (p. 977)	**serialized_type_object_dynamic_allocation_threshold** (p. 995)
	<<***extension***>> *(p. 213)* A threshold, in bytes, for dynamic memory allocation for the serialized TypeObject v1.
DDS_DomainParticipantResourceLimitsQosPolicy (p. 977)	**contentfilter_property_max_length** (p. 997)
	<<***extension***>> *(p. 213)* This field is the maximum length of all data related to a Content-filtered topic.
DDS_DomainParticipantResourceLimitsQosPolicy (p. 977)	**channel_seq_max_length** (p. 997)
	<<***extension***>> *(p. 213)* Maximum number of channels that can be specified in **DDS_MultiChannelQosPolicy** (p. 1214) for MultiChannel DataWriters.
DDS_DomainParticipantResourceLimitsQosPolicy (p. 977)	**channel_filter_expression_max_length** (p. 998)
	<<***extension***>> *(p. 213)* Maximum length of a channel **DDS_ChannelSettings_t::filter_expression** (p. 822) in a MultiChannel DataWriter.
DDS_DomainParticipantResourceLimitsQosPolicy (p. 977)	**participant_property_list_max_length** (p. 998)
	<<***extension***>> *(p. 213)* Maximum number of properties associated with the **DDSDomainParticipant** (p. 1619).
DDS_DomainParticipantResourceLimitsQosPolicy (p. 977)	**participant_property_string_max_length** (p. 998)

	<<***extension***>> *(p. 213)* Maximum string length of the properties associated with the **DDSDomainParticipant** (p. 1619).
DDS_DomainParticipantResourceLimitsQosPolicy (p. 977)	**writer_property_list_max_length** (p. 999)
	<<***extension***>> *(p. 213)* Maximum number of properties associated with a **DDSDataWriter** (p. 1589).
DDS_DomainParticipantResourceLimitsQosPolicy (p. 977)	**writer_property_string_max_length** (p. 999)
	<<***extension***>> *(p. 213)* Maximum string length of the properties associated with a **DDSDataWriter** (p. 1589).
DDS_DomainParticipantResourceLimitsQosPolicy (p. 977)	**reader_property_list_max_length** (p. 999)
	<<***extension***>> *(p. 213)* Maximum number of properties associated with a **DDSDataReader** (p. 1556).
DDS_DomainParticipantResourceLimitsQosPolicy (p. 977)	**reader_property_string_max_length** (p. 1000)
	<<***extension***>> *(p. 213)* Maximum string length of the properties associated with a **DDSDataReader** (p. 1556).
DDS_DomainParticipantResourceLimitsQosPolicy (p. 977)	**max_endpoint_groups** (p. 1000)
	<<***extension***>> *(p. 213)* Maximum number of **DDS_EndpointGroup_t** (p. 1142) allowable in a **DDS_AvailabilityQosPolicy** (p. 805).
DDS_DomainParticipantResourceLimitsQosPolicy (p. 977)	**max_endpoint_group_cumulative_characters** (p. 1000)
	<<***extension***>> *(p. 213)* Maximum number of combined role_name characters allowed in all **DDS_EndpointGroup_t** (p. 1142) in a **DDS_AvailabilityQosPolicy** (p. 805).
DDS_DomainParticipantResourceLimitsQosPolicy (p. 977)	**transport_info_list_max_length** (p. 1001)
	<<***extension***>> *(p. 213)* Maximum number of installed transports to send and receive information about in **DDS_ParticipantBuiltinTopicData::transport_info** (p. 1232).
DDS_DomainParticipantResourceLimitsQosPolicy (p. 977)	**ignored_entity_replacement_kind** (p. 1001)
	[DEPRECATED] (p. **384**) <<***extension***>> *(p. 213)* Replacement policy for the ignored entities. It sets what entity can be replaced when resource limits set in **DDS_DomainParticipantResourceLimitsQosPolicy::ignored_entity_allocation** (p. 984) are reached.

DDS_DomainParticipantResourceLimitsQosPolicy (p. 977)	**remote_topic_query_allocation** (p. 1001)
	<<***extension***>> *(p. 213)* Allocation settings applied to remote TopicQueries.
DDS_DomainParticipantResourceLimitsQosPolicy (p. 977)	**remote_topic_query_hash_buckets** (p. 1002)
	[DEPRECATED] (p. **384**) <<***extension***>> *(p. 213)* Number of hash buckets for remote TopicQueries.
DDS_DomainParticipantResourceLimitsQosPolicy (p. 977)	**writer_data_tag_list_max_length** (p. 1002)
	<<***extension***>> *(p. 213)* Maximum number of data tags associated with a **DDSDataWriter** (p. 1589).
DDS_DomainParticipantResourceLimitsQosPolicy (p. 977)	**writer_data_tag_string_max_length** (p. 1003)
	<<***extension***>> *(p. 213)* Maximum string length of the data tags associated with a **DDSDataWriter** (p. 1589).
DDS_DomainParticipantResourceLimitsQosPolicy (p. 977)	**reader_data_tag_list_max_length** (p. 1003)
	<<***extension***>> *(p. 213)* Maximum number of data tags associated with a **DDSDataReader** (p. 1556).
DDS_DomainParticipantResourceLimitsQosPolicy (p. 977)	**reader_data_tag_string_max_length** (p. 1003)
	<<***extension***>> *(p. 213)* Maximum string length of the data tags associated with a **DDSDataReader** (p. 1556).
DDS_DomainParticipantResourceLimitsQosPolicy (p. 977)	**shmem_ref_transfer_mode_max_segments** (p. 1004)
	<<***extension***>> *(p. 213)* Maximum number of segments created by all DataWriters belonging to a **DDSDomainParticipant** (p. 1619).
DDS_EventQosPolicy (p. 1150)	**thread** (p. 1151)
	<<***extension***>> *(p. 213)* Event thread QoS.
DDS_EventQosPolicy (p. 1150)	**initial_count** (p. 1151)
	<<***extension***>> *(p. 213)* The initial number of events.
DDS_EventQosPolicy (p. 1150)	**max_count** (p. 1152)
	<<***extension***>> *(p. 213)* The maximum number of events.
DDS_DatabaseQosPolicy (p. 831)	**thread** (p. 832)
	<<***extension***>> *(p. 213)* Database thread settings.

DDS_DatabaseQosPolicy (p. 831)	**shutdown_timeout** (p. 832)
	[DEPRECATED] (p. **384**) <<***extension***>> (p. *213*) The maximum wait time during a shutdown.
DDS_DatabaseQosPolicy (p. 831)	**cleanup_period** (p. 833)
	[DEPRECATED] (p. **384**) <<***extension***>> (p. *213*) The period at which the database thread wakes up to remove deleted records.
DDS_DatabaseQosPolicy (p. 831)	**shutdown_cleanup_period** (p. 833)
	[DEPRECATED] (p. **384**) <<***extension***>> (p. *213*) The clean-up period used during database shut-down.
DDS_DatabaseQosPolicy (p. 831)	**initial_records** (p. 833)
	<<***extension***>> (p. *213*) The initial number of total records.
DDS_DatabaseQosPolicy (p. 831)	**max_skiplist_level** (p. 834)
	[DEPRECATED] (p. **384**) <<***extension***>> (p. *213*) The maximum level of the skiplist.
DDS_DatabaseQosPolicy (p. 831)	**initial_weak_references** (p. 835)
	<<***extension***>> (p. *213*) The initial number of weak references.
DDS_DatabaseQosPolicy (p. 831)	**max_weak_references** (p. 834)
	<<***extension***>> (p. *213*) The maximum number of weak references.
DDS_ReceiverPoolQosPolicy (p. 1293)	**thread** (p. 1294)
	<<***extension***>> (p. *213*) Receiver pool thread(s).
DDS_ReceiverPoolQosPolicy (p. 1293)	**buffer_size** (p. 1294)
	<<***extension***>> (p. *213*) The receive buffer size in bytes.
DDS_ReceiverPoolQosPolicy (p. 1293)	**buffer_alignment** (p. 1295)
	[DEPRECATED] (p. **384**) <<***extension***>> (p. *213*) The receive buffer alignment.
DDS_DiscoveryConfigQosPolicy (p. 934)	**participant_liveliness_lease_duration** (p. 937)
	<<***basic***>> (p. *201*) <<***extension***>> (p. *213*) The liveliness lease duration for the participant.
DDS_DiscoveryConfigQosPolicy (p. 934)	**participant_liveliness_assert_period** (p. 937)
	<<***basic***>> (p. *201*) <<***extension***>> (p. *213*) The period to assert liveliness for the participant.
DDS_DiscoveryConfigQosPolicy (p. 934)	**participant_announcement_period** (p. 937)

	<<***basic***>> (p. *201)* <<***extension***>> (p. *213)* The period at which a participant announces itself to potential peers when using the Simple Participant Discovery Protocol 2.0 (SPDP2).
DDS_DiscoveryConfigQosPolicy (p. 934)	**remote_participant_purge_kind** (p. 938)
	[DEPRECATED] (p. **384)** <<***extension***>> (p. *213)* The participant's behavior for maintaining knowledge of remote participants (and their contained entities) with which discovery communication has been lost.
DDS_DiscoveryConfigQosPolicy (p. 934)	**max_liveliness_loss_detection_period** (p. 938)
	[DEPRECATED] (p. **384)** <<***extension***>> (p. *213)* The maximum amount of time between when a remote entity stops maintaining its liveliness and when the matched local entity realizes that fact.
DDS_DiscoveryConfigQosPolicy (p. 934)	**initial_participant_announcements** (p. 939)
	<<***basic***>> (p. *201)* <<***extension***>> (p. *213)* The number of initial announcements sent when a participant is first enabled.
DDS_DiscoveryConfigQosPolicy (p. 934)	**new_remote_participant_announcements** (p. 939)
	<<***basic***>> (p. *201)* <<***extension***>> (p. *213)* The number of participant announcements sent when a remote participant is newly discovered.
DDS_DiscoveryConfigQosPolicy (p. 934)	**min_initial_participant_announcement_period** (p. 940)
	<<***basic***>> (p. *201)* <<***extension***>> (p. *213)* The minimum period between initial announcements when a participant is first enabled or when a remote participant is newly discovered.
DDS_DiscoveryConfigQosPolicy (p. 934)	**max_initial_participant_announcement_period** (p. 940)
	<<***basic***>> (p. *201)* <<***extension***>> (p. *213)* The maximum period between initial announcements when a participant is first enabled or when a remote participant is newly discovered.
DDS_DiscoveryConfigQosPolicy (p. 934)	**participant_reader_resource_limits** (p. 941)
	<<***extension***>> (p. *213)* Resource limits.
DDS_DiscoveryConfigQosPolicy (p. 934)	**publication_reader** (p. 941)
	<<***extension***>> (p. *213)* RTPS protocol-related configuration settings for the RTPS reliable reader associated with a built-in publication reader.
DDS_DiscoveryConfigQosPolicy (p. 934)	**publication_reader_resource_limits** (p. 942)
	<<***extension***>> (p. *213)* Resource limits.

DDS_DiscoveryConfigQosPolicy (p. 934)	**subscription_reader** (p. 942)
	<<***extension***>> *(p. 213)* RTPS protocol-related configuration settings for the RTPS reliable reader associated with a built-in subscription reader.
DDS_DiscoveryConfigQosPolicy (p. 934)	**subscription_reader_resource_limits** (p. 943)
	<<***extension***>> *(p. 213)* Resource limits.
DDS_DiscoveryConfigQosPolicy (p. 934)	**participant_message_writer** (p. 947)
	<<***extension***>> *(p. 213)* RTPS protocol-related configuration settings for the RTPS reliable writer associated with the built-in participant message writer. This parameter only has effect if the matching participant message reader is configured with **DDS_RELIABLE↩ _RELIABILITY_QOS** (p. 633) **DDS_ReliabilityQos↩ PolicyKind** (p. 632).
DDS_DiscoveryConfigQosPolicy (p. 934)	**publication_writer_publish_mode** (p. 948)
	<<***extension***>> *(p. 213)* Publish mode policy for the built-in publication writer.
DDS_DiscoveryConfigQosPolicy (p. 934)	**subscription_writer_publish_mode** (p. 949)
	<<***extension***>> *(p. 213)* Publish mode policy for the built-in subscription writer.
DDS_DiscoveryConfigQosPolicy (p. 934)	**asynchronous_publisher** (p. 949)
	<<***extension***>> *(p. 213)* Asynchronous publishing settings for the discovery **DDSPublisher** (p. 1818) and all entities that are created by it.
DDS_DiscoveryConfigQosPolicy (p. 934)	**default_domain_announcement_period** (p. 949)
	<<***extension***>> *(p. 213)* The period to announce a participant to the default domain 0.
DDS_DiscoveryConfigQosPolicy (p. 934)	**ignore_default_domain_announcements** (p. 950)
	[DEPRECATED] (p. 384) <<***extension***>> *(p. 213)* Used to ignore the announcements received by a participant on the default domain 0 corresponding to participants running on domains IDs other than 0.
DDS_DiscoveryConfigQosPolicy (p. 934)	**participant_message_reader_reliability_kind** (p. 946)
	[DEPRECATED] (p. 384) <<***extension***>> *(p. 213)* Reliability policy for a built-in participant message reader.
DDS_DiscoveryConfigQosPolicy (p. 934)	**participant_message_reader** (p. 947)

	<<***extension***>> *(p. 213)* RTPS reliable reader protocol-related configuration settings for a built-in participant message reader. This parameter only has effect if **DDS_DiscoveryConfigQosPolicy::participant_message_reader_reliability_kind** (p. 946) is set to **DDS_RELIABLE_RELIABILITY_QOS** (p. 633).
DDS_DiscoveryConfigQosPolicy (p. 934)	**publication_writer** (p. 943)
	<<***extension***>> *(p. 213)* RTPS protocol-related configuration settings for the RTPS reliable writer associated with a built-in publication writer.
DDS_DiscoveryConfigQosPolicy (p. 934)	**publication_writer_data_lifecycle** (p. 944)
	<<***extension***>> *(p. 213)* Writer data lifecycle settings for a built-in publication writer.
DDS_DiscoveryConfigQosPolicy (p. 934)	**subscription_writer** (p. 945)
	<<***extension***>> *(p. 213)* RTPS protocol-related configuration settings for the RTPS reliable writer associated with a built-in subscription writer.
DDS_DiscoveryConfigQosPolicy (p. 934)	**subscription_writer_data_lifecycle** (p. 945)
	<<***extension***>> *(p. 213)* Writer data lifecycle settings for a built-in subscription writer.
DDS_DiscoveryConfigQosPolicy (p. 934)	**builtin_discovery_plugins** (p. 946)
	<<***extension***>> *(p. 213)* Mask of built-in discovery plugin kinds.
DDS_DiscoveryConfigQosPolicy (p. 934)	**enabled_builtin_channels** (p. 946)
	<<***extension***>> *(p. 213)* The mask specifying which built-in channels should be enabled.
DDS_DiscoveryConfigQosPolicy (p. 934)	**service_request_writer** (p. 950)
	<<***extension***>> *(p. 213)* RTPS protocol-related configuration settings for the RTPS reliable writer associated with the built-in **DDS_ServiceRequest** (p. 1359) writer.
DDS_DiscoveryConfigQosPolicy (p. 934)	**service_request_writer_data_lifecycle** (p. 951)
	<<***extension***>> *(p. 213)* Writer data lifecycle settings for a built-in **DDS_ServiceRequest** (p. 1359) writer.
DDS_DiscoveryConfigQosPolicy (p. 934)	**service_request_writer_publish_mode** (p. 951)
	<<***extension***>> *(p. 213)* Publish mode policy for the built-in service request writer.
DDS_DiscoveryConfigQosPolicy (p. 934)	**service_request_reader** (p. 952)
	<<***extension***>> *(p. 213)* RTPS reliable reader protocol-related configuration settings for a built-in **DDS_ServiceRequest** (p. 1359) reader.

DDS_DiscoveryConfigQosPolicy (p. 934)	**locator_reachability_lease_duration** (p. 953)
	<<***extension***>> *(p. 213)* The time period after which other DomainParticipants can consider one of their locators as "unreachable" if they do not receive a REACHABILITY PING from this DomainParticipant.
DDS_DiscoveryConfigQosPolicy (p. 934)	**locator_reachability_assert_period** (p. 952)
	<<***extension***>> *(p. 213)* Period at which this DomainParticipant will assert the locators discovered from other DomainParticipants.
DDS_DiscoveryConfigQosPolicy (p. 934)	**locator_reachability_change_detection_period** (p. 953)
	<<***extension***>> *(p. 213)* Period at which this DomainParticipant will check if its locators are reachable from other DomainParticipants.
DDS_DiscoveryConfigQosPolicy (p. 934)	**secure_volatile_writer** (p. 954)
	<<***extension***>> *(p. 213)* RTPS protocol-related configuration settings for the RTPS reliable writer associated with the built-in secure volatile writer.
DDS_DiscoveryConfigQosPolicy (p. 934)	**secure_volatile_writer_publish_mode** (p. 955)
	<<***extension***>> *(p. 213)* Publish mode policy for the built-in secure volatile writer.
DDS_DiscoveryConfigQosPolicy (p. 934)	**secure_volatile_reader** (p. 955)
	<<***extension***>> *(p. 213)* RTPS reliable reader protocol-related configuration settings for the built-in secure volatile reader.
DDS_DiscoveryConfigQosPolicy (p. 934)	**endpoint_type_object_lb_serialization_threshold** (p. 956)
	<<***extension***>> *(p. 213)* Controls when/if the Type↩ Object v1 is compressed when propagated in Simple Endpoint Discovery.
DDS_DiscoveryConfigQosPolicy (p. 934)	**dns_tracker_polling_period** (p. 956)
	<<***extension***>> *(p. 213)* Duration that specifies the period used by the DNS tracker to poll the DNS service and check for changes in the hostnames.
DDS_DiscoveryConfigQosPolicy (p. 934)	**participant_configuration_reader** (p. 959)
	<<***extension***>> *(p. 213)* RTPS protocol-related configuration settings for the RTPS reliable reader associated with a built-in participant configuration reader.
DDS_DiscoveryConfigQosPolicy (p. 934)	**participant_configuration_reader_resource_limits** (p. 959)

	<<***extension***>> *(p. 213)* Resource limits for the built-in topic participant configuration reader.
DDS_DiscoveryConfigQosPolicy (p. 934)	**participant_configuration_writer_publish_mode** (p. 957)
	<<***extension***>> *(p. 213)* Publish mode policy for the built-in participant configuration writer.
DDS_DiscoveryConfigQosPolicy (p. 934)	**participant_configuration_writer** (p. 957)
	<<***extension***>> *(p. 213)* RTPS protocol-related configuration settings for the RTPS reliable writer associated with a built-in participant configuration writer.
DDS_DiscoveryConfigQosPolicy (p. 934)	**participant_configuration_writer_data_lifecycle** (p. 958)
	<<***extension***>> *(p. 213)* Writer data lifecycle settings for a built-in participant configuration writer.
DDS_DiscoveryConfigQosPolicy (p. 934)	**request_types_filter** (p. 959)
	<<***extension***>> *(p. 213)* Topic name pattern for requesting all types via Type Lookup Service.
DDS_RtpsWellKnownPorts_t (p. 1338)	**port_base** (p. 1340)
	<<***extension***>> *(p. 213)* The base port offset.
DDS_RtpsWellKnownPorts_t (p. 1338)	**domain_id_gain** (p. 1341)
	<<***extension***>> *(p. 213)* Tunable domain gain parameter.
DDS_RtpsWellKnownPorts_t (p. 1338)	**participant_id_gain** (p. 1342)
	<<***extension***>> *(p. 213)* Tunable participant gain parameter.
DDS_RtpsWellKnownPorts_t (p. 1338)	**builtin_multicast_port_offset** (p. 1342)
	<<***extension***>> *(p. 213)* Additional offset for **metatraffic** multicast port.
DDS_RtpsWellKnownPorts_t (p. 1338)	**builtin_unicast_port_offset** (p. 1342)
	<<***extension***>> *(p. 213)* Additional offset for **metatraffic** unicast port.
DDS_RtpsWellKnownPorts_t (p. 1338)	**user_multicast_port_offset** (p. 1343)
	<<***extension***>> *(p. 213)* Additional offset for **usertraffic** multicast port.
DDS_RtpsWellKnownPorts_t (p. 1338)	**user_unicast_port_offset** (p. 1343)
	<<***extension***>> *(p. 213)* Additional offset for **usertraffic** unicast port.
DDS_RtpsReliableReaderProtocol_t (p. 1314)	**min_heartbeat_response_delay** (p. 1315)

	<<***extension***>> *(*p. *213)* The minimum delay to respond to a heartbeat.
DDS_RtpsReliableReaderProtocol_t (p. 1314)	**max_heartbeat_response_delay** (p. 1315)
	<<***extension***>> *(*p. *213)* The maximum delay to respond to a heartbeat.
DDS_RtpsReliableReaderProtocol_t (p. 1314)	**heartbeat_suppression_duration** (p. 1316)
	<<***extension***>> *(*p. *213)* The duration a reader ignores consecutively received heartbeats
DDS_RtpsReliableReaderProtocol_t (p. 1314)	**nack_period** (p. 1316)
	<<***extension***>> *(*p. *213)* The period at which to send NACKs
DDS_RtpsReliableReaderProtocol_t (p. 1314)	**receive_window_size** (p. 1317)
	<<***extension***>> *(*p. *213)* The number of received out-of-order samples a reader can keep at a time
DDS_RtpsReliableReaderProtocol_t (p. 1314)	**round_trip_time** (p. 1317)
	<<***extension***>> *(*p. *213)* The duration from sending a NACK to receiving a repair of a sample
DDS_RtpsReliableReaderProtocol_t (p. 1314)	**app_ack_period** (p. 1318)
	<<***extension***>> *(*p. *213)* The period at which application-level acknowledgment messages are sent
DDS_RtpsReliableReaderProtocol_t (p. 1314)	**min_app_ack_response_keep_duration** (p. 1318)
	<<***extension***>> *(*p. *213)* Minimum duration for which application-level acknowledgment response data is kept
DDS_RtpsReliableReaderProtocol_t (p. 1314)	**samples_per_app_ack** (p. 1319)
	<<***extension***>> *(*p. *213)* The minimum number of samples acknowledged by one application-level acknowledgment message
DDS_RtpsReliableWriterProtocol_t (p. 1320)	**low_watermark** (p. 1321)
	<<***extension***>> *(*p. *213)* When the number of unacknowledged samples in the current send window of a reliable writer meets or falls below this threshold, the **DDS_RELIABLE_WRITER_CACHE_CHANGED↩ _STATUS** (p. 545) is considered to have changed.
DDS_RtpsReliableWriterProtocol_t (p. 1320)	**high_watermark** (p. 1322)
	<<***extension***>> *(*p. *213)* When the number of unacknowledged samples in the current send window of a reliable writer meets or exceeds this threshold, the **DDS_RELIABLE_WRITER_CACHE_CHANGED↩ _STATUS** (p. 545) is considered to have changed.

DDS_RtpsReliableWriterProtocol_t (p. 1320)	**heartbeat_period** (p. 1323)
	<<***basic***>> *(p. 201)* <<***extension***>> *(p. 213)* The period at which to send heartbeats.
DDS_RtpsReliableWriterProtocol_t (p. 1320)	**fast_heartbeat_period** (p. 1323)
	<<***basic***>> *(p. 201)* <<***extension***>> *(p. 213)* An alternative heartbeat period used when a reliable writer needs to flush its unacknowledged samples more quickly.
DDS_RtpsReliableWriterProtocol_t (p. 1320)	**late_joiner_heartbeat_period** (p. 1324)
	<<***basic***>> *(p. 201)* <<***extension***>> *(p. 213)* An alternative heartbeat period used when a reliable reader joins late and needs to be caught up on cached samples of a reliable writer more quickly than the normal heartbeat rate.
DDS_RtpsReliableWriterProtocol_t (p. 1320)	**virtual_heartbeat_period** (p. 1325)
	<<***extension***>> *(p. 213)* The period at which to send virtual heartbeats. Virtual heartbeats inform the reliable reader about the range of samples currently present, for each virtual GUID, in the reliable writer's queue.
DDS_RtpsReliableWriterProtocol_t (p. 1320)	**samples_per_virtual_heartbeat** (p. 1325)
	<<***extension***>> *(p. 213)* The number of samples that a reliable writer has to publish before sending a virtual heartbeat.
DDS_RtpsReliableWriterProtocol_t (p. 1320)	**max_heartbeat_retries** (p. 1326)
	<<***extension***>> *(p. 213)* The maximum number of *periodic* heartbeat retries before marking a remote reader as inactive.
DDS_RtpsReliableWriterProtocol_t (p. 1320)	**inactivate_nonprogressing_readers** (p. 1326)
	<<***extension***>> *(p. 213)* Whether to treat remote readers as inactive when their NACKs do not progress
DDS_RtpsReliableWriterProtocol_t (p. 1320)	**heartbeats_per_max_samples** (p. 1327)
	<<***extension***>> *(p. 213)* The number of piggyback heartbeats sent per max send window.
DDS_RtpsReliableWriterProtocol_t (p. 1320)	**min_nack_response_delay** (p. 1328)
	<<***extension***>> *(p. 213)* The minimum delay to respond to a NACK or NACK_FRAG.
DDS_RtpsReliableWriterProtocol_t (p. 1320)	**max_nack_response_delay** (p. 1328)
	<<***extension***>> *(p. 213)* The maximum delay to respond to a NACK or NACK_FRAG.
DDS_RtpsReliableWriterProtocol_t (p. 1320)	**nack_suppression_duration** (p. 1329)

	<<***extension***>> *(p. 213)* The duration for ignoring consecutive NACKs/NACK_FRAGs that may trigger redundant repairs.
DDS_RtpsReliableWriterProtocol_t (p. 1320)	**max_bytes_per_nack_response** (p. 1329)
	<<***extension***>> *(p. 213)* The maximum total message size when resending rejected samples or data fragments.
DDS_RtpsReliableWriterProtocol_t (p. 1320)	**disable_positive_acks_enable_adaptive_sample_keep_duration** (p. 1331)
	[DEPRECATED] (p. 384) <<***extension***>> *(p. 213)* Enables dynamic adjustment of sample keep duration in response to congestion.
DDS_RtpsReliableWriterProtocol_t (p. 1320)	**disable_positive_acks_min_sample_keep_duration** (p. 1330)
	<<***extension***>> *(p. 213)* The minimum duration a sample is queued for ACK-disabled readers.
DDS_RtpsReliableWriterProtocol_t (p. 1320)	**disable_positive_acks_max_sample_keep_duration** (p. 1331)
	<<***extension***>> *(p. 213)* The maximum duration a sample is queued for ACK-disabled readers.
DDS_RtpsReliableWriterProtocol_t (p. 1320)	**disable_positive_acks_decrease_sample_keep_duration_factor** (p. 1332)
	[DEPRECATED] (p. 384) <<***extension***>> *(p. 213)* Controls rate of contraction of dynamic sample keep duration.
DDS_RtpsReliableWriterProtocol_t (p. 1320)	**disable_positive_acks_increase_sample_keep_duration_factor** (p. 1332)
	[DEPRECATED] (p. 384) <<***extension***>> *(p. 213)* Controls rate of growth of dynamic sample keep duration.
DDS_RtpsReliableWriterProtocol_t (p. 1320)	**min_send_window_size** (p. 1333)
	<<***basic***>> *(p. 201)* <<***extension***>> *(p. 213)* Minimum size of send window of unacknowledged samples
DDS_RtpsReliableWriterProtocol_t (p. 1320)	**max_send_window_size** (p. 1333)
	<<***basic***>> *(p. 201)* <<***extension***>> *(p. 213)* Maximum size of send window of unacknowledged samples
DDS_RtpsReliableWriterProtocol_t (p. 1320)	**send_window_update_period** (p. 1334)
	<<***extension***>> *(p. 213)* Period in which send window may be dynamically changed
DDS_RtpsReliableWriterProtocol_t (p. 1320)	**send_window_increase_factor** (p. 1335)

	<<***extension***>> *(p. 213)* Increases send window size by this percentage when reacting dynamically to network conditions
DDS_RtpsReliableWriterProtocol_t (p. 1320)	**send_window_decrease_factor** (p. 1336)
	<<***extension***>> *(p. 213)* Decreases send window size by this percentage when reacting dynamically to network conditions
DDS_RtpsReliableWriterProtocol_t (p. 1320)	**enable_multicast_periodic_heartbeat** (p. 1337)
	<<***extension***>> *(p. 213)* Whether periodic heartbeat messages are sent over multicast
DDS_RtpsReliableWriterProtocol_t (p. 1320)	**multicast_resend_threshold** (p. 1337)
	<<***extension***>> *(p. 213)* The minimum number of requesting readers needed to trigger a multicast resend
DDS_RtpsReliableWriterProtocol_t (p. 1320)	**disable_repair_piggyback_heartbeat** (p. 1338)
	<<***extension***>> *(p. 213)* Prevents piggyback heartbeats from being sent with repair samples
DDS_BuiltinTopicReaderResourceLimits_t (p. 815)	**initial_samples** (p. 816)
	<<***extension***>> *(p. 213)* Initial number of samples.
DDS_BuiltinTopicReaderResourceLimits_t (p. 815)	**max_samples** (p. 816)
	<<***extension***>> *(p. 213)* Maximum number of samples.
DDS_BuiltinTopicReaderResourceLimits_t (p. 815)	**initial_infos** (p. 817)
	<<***extension***>> *(p. 213)* Initial number of sample infos.
DDS_BuiltinTopicReaderResourceLimits_t (p. 815)	**max_infos** (p. 817)
	<<***extension***>> *(p. 213)* Maximum number of sample infos.
DDS_BuiltinTopicReaderResourceLimits_t (p. 815)	**initial_outstanding_reads** (p. 817)
	<<***extension***>> *(p. 213)* The initial number of outstanding reads that have not called finish yet on the same built-in topic **DDSDataReader** (p. 1556).
DDS_BuiltinTopicReaderResourceLimits_t (p. 815)	**max_outstanding_reads** (p. 818)
	<<***extension***>> *(p. 213)* The maximum number of outstanding reads that have not called finish yet on the same built-in topic **DDSDataReader** (p. 1556).
DDS_BuiltinTopicReaderResourceLimits_t (p. 815)	**max_samples_per_read** (p. 818)
	<<***extension***>> *(p. 213)* Maximum number of samples that can be read/taken on a same built-in topic **DDSDataReader** (p. 1556).

DDS_BuiltinTopicReaderResourceLimits_t (p. 815)	**disable_fragmentation_support** (p. 818)
	[DEPRECATED] (p. **384)** <<***extension***>> *(*p. *213)* Determines whether the built-in topic **DDSDataReader** (p. 1556) can receive fragmented samples.
DDS_BuiltinTopicReaderResourceLimits_t (p. 815)	**max_fragmented_samples** (p. 819)
	<<***extension***>> *(*p. *213)* The maximum number of samples for which the built-in topic **DDSDataReader** (p. 1556) may store fragments at a given point in time.
DDS_BuiltinTopicReaderResourceLimits_t (p. 815)	**initial_fragmented_samples** (p. 819)
	<<***extension***>> *(*p. *213)* The initial number of samples for which a built-in topic **DDSDataReader** (p. 1556) may store fragments.
DDS_BuiltinTopicReaderResourceLimits_t (p. 815)	**max_fragmented_samples_per_remote_writer** (p. 820)
	<<***extension***>> *(*p. *213)* The maximum number of samples per remote writer for which a built-in topic **DDSDataReader** (p. 1556) may store fragments.
DDS_BuiltinTopicReaderResourceLimits_t (p. 815)	**max_fragments_per_sample** (p. 820)
	<<***extension***>> *(*p. *213)* Maximum number of fragments for a single sample.
DDS_BuiltinTopicReaderResourceLimits_t (p. 815)	**dynamically_allocate_fragmented_samples** (p. 821)
	<<***extension***>> *(*p. *213)* Determines whether the built-in topic **DDSDataReader** (p. 1556) pre-allocates storage for storing fragmented samples.
DDS_TransportUnicastSettings_t (p. 1424)	**transports** (p. 1425)
	<<***extension***>> *(*p. *213)* A sequence of transport aliases that specifies the unicast interfaces on which to receive *unicast* traffic for the entity.
DDS_TransportUnicastSettings_t (p. 1424)	**receive_port** (p. 1425)
	<<***extension***>> *(*p. *213)* The unicast port on which the entity can receive data.
DDS_TransportMulticastSettings_t (p. 1417)	**transports** (p. 1418)
	<<***extension***>> *(*p. *213)* A sequence of transport aliases that specifies the transports on which to receive *multicast* traffic for the entity.
DDS_TransportMulticastSettings_t (p. 1417)	**receive_address** (p. 1418)
	<<***extension***>> *(*p. *213)* The multicast group address on which the entity can receive data.
DDS_TransportMulticastSettings_t (p. 1417)	**receive_port** (p. 1418)

	<<***extension***>> *(p. 213)* The multicast port on which the entity can receive data.
DDS_TransportMulticastMapping_t (p. 1410)	**addresses** (p. 1411)
	<<***extension***>> *(p. 213)* A string containing a comma-separated list of IP addresses or IP address ranges to be used to receive *multicast* traffic for the entity with a topic that matches the **DDS_TransportMulticastMapping_↩ t::topic_expression** (p. 1411).
DDS_TransportMulticastMapping_t (p. 1410)	**topic_expression** (p. 1411)
	<<***extension***>> *(p. 213)* A regular expression that will be used to map topic names to corresponding multicast receive addresses.
DDS_TransportMulticastMapping_t (p. 1410)	**mapping_function** (p. 1412)
	<<***extension***>> *(p. 213)* Specifies a function that will define the mapping between a topic name and a specific multicast address from a list of addresses.
DDS_PropertyQosPolicy (p. 1260)	**value** (p. 1262)
	<<***basic***>> *(p. 201)* <<***extension***>> *(p. 213)* Sequence of properties.
DDS_EntityNameQosPolicy (p. 1147)	**name** (p. 1148)
	<<***basic***>> *(p. 201)* <<***extension***>> *(p. 213)* The name of the entity.
DDS_EntityNameQosPolicy (p. 1147)	**role_name** (p. 1148)
	<<***extension***>> *(p. 213)* The entity role name.
DDS_ServiceQosPolicy (p. 1358)	**kind** (p. 1359)
	<<***extension***>> *(p. 213)* The kind of service.
DDS_BatchQosPolicy (p. 810)	**enable** (p. 810)
	<<***basic***>> *(p. 201)* <<***extension***>> *(p. 213)* Specifies whether or not batching is enabled.
DDS_BatchQosPolicy (p. 810)	**max_data_bytes** (p. 811)
	<<***basic***>> *(p. 201)* <<***extension***>> *(p. 213)* The maximum cumulative length of all serialized samples in a batch.
DDS_BatchQosPolicy (p. 810)	**max_samples** (p. 811)
	<<***basic***>> *(p. 201)* <<***extension***>> *(p. 213)* The maximum number of samples in a batch.
DDS_BatchQosPolicy (p. 810)	**max_flush_delay** (p. 812)
	<<***extension***>> *(p. 213)* The maximum flush delay.

DDS_BatchQosPolicy (p. 810)	**source_timestamp_resolution** (p. 812)
	<<***extension***>> *(p. 213)* Batch source timestamp resolution.
DDS_BatchQosPolicy (p. 810)	**thread_safe_write** (p. 813)
	<<***extension***>> *(p. 213)* Determines whether or not the write operation is thread safe.
DDS_ProfileQosPolicy (p. 1257)	**string_profile** (p. 1258)
	<<***basic***>> *(p. 201)* <<***extension***>> *(p. 213)* Sequence of strings containing a XML document to load.
DDS_ProfileQosPolicy (p. 1257)	**url_profile** (p. 1258)
	<<***basic***>> *(p. 201)* <<***extension***>> *(p. 213)* Sequence of **URL groups** (p. 392) containing a set of XML documents to load.
DDS_ProfileQosPolicy (p. 1257)	**ignore_user_profile** (p. 1258)
	<<***basic***>> *(p. 201)* <<***extension***>> *(p. 213)* Ignores the file USER_QOS_PROFILES.xml in the current working directory.
DDS_ProfileQosPolicy (p. 1257)	**ignore_environment_profile** (p. 1258)
	<<***basic***>> *(p. 201)* <<***extension***>> *(p. 213)* Ignores the value of the **NDDS_QOS_PROFILES** (p. 392).
DDS_ProfileQosPolicy (p. 1257)	**ignore_resource_profile** (p. 1259)
	<<***basic***>> *(p. 201)* <<***extension***>> *(p. 213)* Ignores the file NDDS_QOS_PROFILES.xml.
DDS_LoggingQosPolicy (p. 1188)	**verbosity** (p. 1189)
	<<***basic***>> *(p. 201)* <<***extension***>> *(p. 213)* The verbosities at which RTI Connext diagnostic information is logged.
DDS_LoggingQosPolicy (p. 1188)	**category** (p. 1189)
	<<***basic***>> *(p. 201)* <<***extension***>> *(p. 213)* Categories of logged messages.
DDS_LoggingQosPolicy (p. 1188)	**print_format** (p. 1189)
	<<***extension***>> *(p. 213)* The format used to output RTI Connext diagnostic information.
DDS_LoggingQosPolicy (p. 1188)	**output_file** (p. 1189)
	<<***basic***>> *(p. 201)* <<***extension***>> *(p. 213)* Specifies the file to which log messages will be redirected to.
DDS_LoggingQosPolicy (p. 1188)	**output_file_suffix** (p. 1190)
	<<***basic***>> *(p. 201)* <<***extension***>> *(p. 213)* Sets the file suffix when logging to a set of files

DDS_LoggingQosPolicy (p. 1188)	**max_bytes_per_file** (p. 1190)
	<<***extension***>> *(p. 213)* Specifies the maximum number of bytes a single file can contain
DDS_LoggingQosPolicy (p. 1188)	**max_files** (p. 1191)
	<<***extension***>> *(p. 213)* Specifies the maximum number of files to create before overwritting the previous ones.
DDS_MonitoringEventDistributionSettings (p. 1197)	**concurrency_level** (p. 1198)
	<<***extension***>> *(p. 213)* Defines how concurrent the push of event metrics to RTI Monitoring Library 2.0 is.
DDS_MonitoringEventDistributionSettings (p. 1197)	**datawriter_qos_profile_name** (p. 1199)
	<<***extension***>> *(p. 213)* The fully qualified name of the profile used to configure the **DDSDataWriter** (p. 1589) that distributes event metrics.
DDS_MonitoringEventDistributionSettings (p. 1197)	**thread** (p. 1199)
	<<***extension***>> *(p. 213)* The settings of the event metric thread.
DDS_MonitoringEventDistributionSettings (p. 1197)	**publication_period** (p. 1200)
	<<***basic***>> *(p. 201)* <<***extension***>> *(p. 213)* Period at which the event metric thread publishes the event metrics that have changed since the last time they were published.
DDS_MonitoringPeriodicDistributionSettings (p. 1209)	**datawriter_qos_profile_name** (p. 1209)
	<<***extension***>> *(p. 213)* The fully qualified name of the profile used to configure the **DDSDataWriter** (p. 1589) that distributes periodic metrics.
DDS_MonitoringPeriodicDistributionSettings (p. 1209)	**thread** (p. 1209)
	<<***extension***>> *(p. 213)* The settings of the periodic metric thread.
DDS_MonitoringPeriodicDistributionSettings (p. 1209)	**polling_period** (p. 1210)
	<<***basic***>> *(p. 201)* <<***extension***>> *(p. 213)* Period at which the periodic metric thread polls and publishes the periodic metrics.
DDS_MonitoringLoggingDistributionSettings (p. 1200)	**concurrency_level** (p. 1201)
	<<***extension***>> *(p. 213)* Defines how concurrent the push of log messages to RTI Monitoring Library 2.0 is.

DDS_MonitoringLoggingDistributionSettings (p. 1200)	**max_historical_logs** (p. 1202)
	<<***basic***>> *(p. 201)* <<***extension***>> *(p. 213)* The number of log messages that RTI Monitoring Library 2.0 will keep as history.
DDS_MonitoringLoggingDistributionSettings (p. 1200)	**datawriter_qos_profile_name** (p. 1202)
	<<***extension***>> *(p. 213)* The fully qualified name of the profile used to configure the **DDSDataWriter** (p. 1589) that distributes log messages.
DDS_MonitoringLoggingDistributionSettings (p. 1200)	**thread** (p. 1203)
	<<***extension***>> *(p. 213)* The settings of the logging thread.
DDS_MonitoringLoggingDistributionSettings (p. 1200)	**publication_period** (p. 1203)
	<<***basic***>> *(p. 201)* <<***extension***>> *(p. 213)* Period at which the logging thread publishes log messages.
DDS_MonitoringDedicatedParticipantSettings (p. 1193)	**enable** (p. 1193)
	<<***basic***>> *(p. 201)* <<***extension***>> *(p. 213)* Enables the use of a dedicated **DDSDomainParticipant** (p. 1619) to distribute the RTI Connext application telemetry data.
DDS_MonitoringDedicatedParticipantSettings (p. 1193)	**domain_id** (p. 1194)
	<<***basic***>> *(p. 201)* <<***extension***>> *(p. 213)* The domain ID used in the creation of RTI Monitoring Library 2.0 **DDSDomainParticipant** (p. 1619).
DDS_MonitoringDedicatedParticipantSettings (p. 1193)	**participant_qos_profile_name** (p. 1194)
	<<***extension***>> *(p. 213)* The fully qualified name of the profile used to configure the **DDSDomain↩ Participant** (p. 1619) that will be used to distribute telemetry data.
DDS_MonitoringDedicatedParticipantSettings (p. 1193)	**collector_initial_peers** (p. 1194)
	<<***basic***>> *(p. 201)* <<***extension***>> *(p. 213)* Determines the initial list of peers that the discovery process will contact to send announcements about the presence of the **DDS_MonitoringDistributionSettings↩ ::dedicated_participant** (p. 1196).
DDS_MonitoringDistributionSettings (p. 1195)	**dedicated_participant** (p. 1196)

	<<***extension***>> *(p. 213)* Configures the use of a dedicated **DDSDomainParticipant** (p. 1619) to distribute the RTI Connext application telemetry data.
DDS_MonitoringDistributionSettings (p. 1195)	**publisher_qos_profile_name** (p. 1196)
	<<***extension***>> *(p. 213)* The fully qualified name of the profile used to configure the Publishers that distribute telemetry data.
DDS_MonitoringDistributionSettings (p. 1195)	**event_settings** (p. 1196)
	<<***extension***>> *(p. 213)* Configures the distribution of event metrics.
DDS_MonitoringDistributionSettings (p. 1195)	**periodic_settings** (p. 1197)
	<<***extension***>> *(p. 213)* Configures the distribution of periodic metrics.
DDS_MonitoringDistributionSettings (p. 1195)	**logging_settings** (p. 1197)
	<<***extension***>> *(p. 213)* Configures the distribution of logging messages.
DDS_MonitoringMetricSelection (p. 1206)	**resource_selection** (p. 1206)
	<<***extension***>> *(p. 213)* An expression pattern that selects a subset of resources by matching the pattern to the resource names. **DDS_MonitoringMetric↩ Selection::enabled_metrics_selection** (p. 1207) and **DDS_MonitoringMetricSelection::disabled_↩ metrics_selection** (p. 1207) are applied to this subset of resources.
DDS_MonitoringMetricSelection (p. 1206)	**enabled_metrics_selection** (p. 1207)
	<<***extension***>> *(p. 213)* A sequence of POSIX fnmatch patterns that match the names of the metrics that should be collected and distributed for the observable resources selected by **DDS_MonitoringMetric↩ Selection::resource_selection** (p. 1206).
DDS_MonitoringMetricSelection (p. 1206)	**disabled_metrics_selection** (p. 1207)
	<<***extension***>> *(p. 213)* A sequence of POSIX fnmatch patterns that mach the names of the metrics that should not be collected and distributed for the observable resources selected by **DDS_MonitoringMetric↩ Selection::resource_selection** (p. 1206).
DDS_MonitoringLoggingForwardingSettings (p. 1204)	**middleware_forwarding_level** (p. 1204)
	<<***basic***>> *(p. 201)* <<***extension***>> *(p. 213)* Log messages with **NDDS_CONFIG_LOG_FACILITY_↩ MIDDLEWARE** (p. 727) and a log level more verbose than this value will not be distributed by RTI Monitoring Library 2.0.

DDS_MonitoringLoggingForwardingSettings (p. 1204)	**security_event_forwarding_level** (p. 1204)
	*<<**basic**>> (p. 201) <<**extension**>> (p. 213)* Log messages with **NDDS_CONFIG_LOG_FACILITY_↩ SECURITY_EVENT** (p. 727) and a log level more verbose than this value will not be distributed by RTI Monitoring Library 2.0.
DDS_MonitoringLoggingForwardingSettings (p. 1204)	**service_forwarding_level** (p. 1205)
	*<<**basic**>> (p. 201) <<**extension**>> (p. 213)* Log messages with **NDDS_CONFIG_LOG_FACILITY_↩ SERVICE** (p. 727) and a log level more verbose than this value will not be distributed by RTI Monitoring Library 2.0.
DDS_MonitoringLoggingForwardingSettings (p. 1204)	**user_forwarding_level** (p. 1205)
	*<<**basic**>> (p. 201) <<**extension**>> (p. 213)* Log messages with **NDDS_CONFIG_LOG_FACILITY_↩ USER** (p. 727) and a log level more verbose than this value will not be distributed by RTI Monitoring Library 2.0.
DDS_MonitoringTelemetryData (p. 1213)	**metrics** (p. 1213)
	*<<**basic**>> (p. 201) <<**extension**>> (p. 213)* Sequence of **DDS_MonitoringMetricSelection** (p. 1206) containing the event and periodic metrics that will be collected and distributed for a given set of observable resources.
DDS_MonitoringTelemetryData (p. 1213)	**logs** (p. 1213)
	*<<**basic**>> (p. 201) <<**extension**>> (p. 213)* **DDS_MonitoringLoggingForwardingSettings** (p. 1204) containing the **NDDS_Config_Syslog↩ Verbosity** (p. 727) levels that will be forwarded for the different **NDDS_Config_LogFacility** (p. 727).
DDS_MonitoringQosPolicy (p. 1210)	**enable** (p. 1211)
	*<<**basic**>> (p. 201) <<**extension**>> (p. 213)* Enables the collection and distribution of telemetry data for an RTI Connext application using RTI Monitoring Library 2.0.
DDS_MonitoringQosPolicy (p. 1210)	**application_name** (p. 1211)
	*<<**basic**>> (p. 201) <<**extension**>> (p. 213)* The name of the resource that represents this RTI Connext application.
DDS_MonitoringQosPolicy (p. 1210)	**distribution_settings** (p. 1212)

	<<***extension***>> *(p. 213)* Configures the distribution of telemetry data.
DDS_MonitoringQosPolicy (p. 1210)	**telemetry_data** (p. 1212)
	<<***extension***>> *(p. 213)* Configures the telemetry data that will be distributed.
DDS_LocatorFilter_t (p. 1185)	**locators** (p. 1185)
	<<***extension***>> *(p. 213)* Sequence containing from one to 16 **DDS_Locator_t** (p. 1184), used to specify the multicast address locators of an individual channel within a MultiChannel DataWriter.
DDS_LocatorFilter_t (p. 1185)	**filter_expression** (p. 1185)
	<<***extension***>> *(p. 213)* A logical expression used to determine the data that will be published in the channel.
DDS_LocatorFilterQosPolicy (p. 1186)	**locator_filters** (p. 1187)
	<<***extension***>> *(p. 213)* A sequence of **DDS_LocatorFilter_t** (p. 1185). Each **DDS_LocatorFilter_t** (p. 1185) reports the configuration of a single channel of a MultiChannel DataWriter.
DDS_LocatorFilterQosPolicy (p. 1186)	**filter_name** (p. 1187)
	<<***extension***>> *(p. 213)* Name of the filter class used to describe the filter expressions of a MultiChannel DataWriter.
DDS_ChannelSettings_t (p. 822)	**multicast_settings** (p. 822)
	<<***extension***>> *(p. 213)* A sequence of **DDS_TransportMulticastSettings_t** (p. 1417) used to configure the multicast addresses associated with a channel.
DDS_ChannelSettings_t (p. 822)	**filter_expression** (p. 822)
	<<***extension***>> *(p. 213)* A logical expression used to determine the data that will be published in the channel.
DDS_ChannelSettings_t (p. 822)	**priority** (p. 823)
	<<***extension***>> *(p. 213)* Publication priority
DDS_MultiChannelQosPolicy (p. 1214)	**channels** (p. 1215)
	<<***extension***>> *(p. 213)* A sequence of **DDS_ChannelSettings_t** (p. 822) used to configure the channels' properties. If the length of the sequence is zero, the QoS policy will be ignored.
DDS_MultiChannelQosPolicy (p. 1214)	**filter_name** (p. 1215)
	<<***extension***>> *(p. 213)* Name of the filter class used to describe the filter expressions of a MultiChannel DataWriter.

DDS_TopicQueryDispatchQosPolicy (p. 1404)	**enable** (p. 1405)
	<<***extension***>> *(p. 213)* Allows this writer to dispatch TopicQueries
DDS_TopicQueryDispatchQosPolicy (p. 1404)	**publication_period** (p. 1405)
	<<***basic***>> *(p. 201)* <<***extension***>> *(p. 213)* Sets the periodic interval at which samples are published.
DDS_TopicQueryDispatchQosPolicy (p. 1404)	**samples_per_period** (p. 1405)
	<<***extension***>> *(p. 213)* Sets the maximum number of samples to publish in each publication_period.

7.50 Functional Categories

These categories help you to discover which QoS may be relevant or related to a desired functionality or feature. For example, if you are interested in **Sending Large Data** (p. 290), then looking at any QoS in that category will be helpful.

Modules

- **Application Acknowledgement**

  This page lists all QoS related to application acknowledgement.
- **Application-Defined Metadata**

  This page lists QoS fields that can be used to send application-specific data during participant and endpoint discovery.
- **Application Startup**

  Use these QoS fields to configure entity creation and application startup settings.
- **Batching**

  Use these QoS fields to configure the batching feature.
- **Concurrency**

  Use these QoS fields to configure threads and concurrency settings in RTI Connext.
- **Content Filtering**

  Use these QoS fields if you are using content filtering on your data.
- **Data Availability**

  Use these QoS fields to configure how much data, which data, and when that data is available to an application.
- **Debuggability**

  Use these QoS to configure logging and observability capabilities to help debug RTI Connext applications.
- **Discovery**

  Use these QoS fields to configure how applications discover each other.
- **Discovery Isolation**

  Use these QoS fields to isolate applications from each other.
- **Durability**

  Use these QoS fields to configure how much and how long data remains retrievable in the system.
- **Flow Control**

  Use these QoS fields to configure features which help to control the rate at which data is sent, preventing network congestion.

- **Fragmentation**

  QoS related to DDS-level data fragmentation.
- **Instance Lifecycle**

  QoS that control the lifecycle of instances.
- **Sending Large Data**

  Use these QoS fields to optimize the process of sending large data.
- **Liveliness**

  Use these QoS fields to manage the liveliness of participants and endpoints, ensuring active communication and timely updates of the system state.
- **Logging**

  Use these QoS fields to configure the RTI Connext logging infrastructure.
- **Middleware Memory Management**

  Use these QoS fields to configure the memory usage of the middleware, including various initial and max limits.
- **QoS Profile Management**

  Use these QoS fields to configure how QoS profiles are loaded and which ones should be used when entities are created.
- **Reliability QoS**

  Reliability related QoS.
- **Sample Memory Management QoS**

  Use these QoS fields to configure how Connext manages memory related to the samples that are stored and managed by the DataReader and DataWriter queues.
- **Sample Ordering**

  Use these QoS fields to configure how samples are ordered when stored and accessed by a DataReader.
- **Security**

  Use these QoS fields to enable and configure some security-related capabilities in RTI Connext.
- **Throughput**

  Use these QoS fields to configure features which may be helpful to increase throughput.
- **Transports**

  Use these QoS fields to configure aspects of the transports (udp, shared memory, etc.) used for communication.
- **Topic Queries**

  Use these QoS fields to configure the topic query feature.
- **Type System**

  Use these QoS fields to configure aspects of the type system.

7.50.1 Detailed Description

These categories help you to discover which QoS may be relevant or related to a desired functionality or feature. For example, if you are interested in **Sending Large Data** (p. 290), then looking at any QoS in that category will be helpful.

7.51 Application Acknowledgement

This page lists all QoS related to application acknowledgement.

This page lists all QoS related to application acknowledgement.

See `Application Acknowledgment`, in the *Core Libraries User's Manual*, for more information about this feature.

Advanced Settings

DDS_ReliabilityQosPolicy (p. 1295)	**acknowledgment_kind** (p. 1298)
	<<***extension***>> *(p. 213)* Kind of reliable acknowledgment
DDS_DataReaderResourceLimitsQosPolicy (p. 867)	**max_app_ack_response_length** (p. 881)
	<<***extension***>> *(p. 213)* Maximum length of application-level acknowledgment response data
DDS_DataWriterResourceLimitsQosPolicy (p. 917)	**max_app_ack_remote_readers** (p. 924)
	<<***extension***>> *(p. 213)* The maximum number of application-level acknowledging remote readers supported by a **DDSDataWriter** (p. 1589)
DDS_DataWriterProtocolQosPolicy (p. 892)	**propagate_app_ack_with_no_response** (p. 896)
	<<***extension***>> *(p. 213)* Controls whether or not a **DDSDataWriter** (p. 1589) receives **DDSDataWriter↩ Listener::on_application_acknowledgment** (p. 1617) notifications with an empty or invalid response.
DDS_RtpsReliableReaderProtocol_t (p. 1314)	**app_ack_period** (p. 1318)
	<<***extension***>> *(p. 213)* The period at which application-level acknowledgment messages are sent
DDS_RtpsReliableReaderProtocol_t (p. 1314)	**min_app_ack_response_keep_duration** (p. 1318)
	<<***extension***>> *(p. 213)* Minimum duration for which application-level acknowledgment response data is kept
DDS_RtpsReliableReaderProtocol_t (p. 1314)	**samples_per_app_ack** (p. 1319)
	<<***extension***>> *(p. 213)* The minimum number of samples acknowledged by one application-level acknowledgment message

7.52 Application-Defined Metadata

This page lists QoS fields that can be used to send application-specific data during participant and endpoint discovery.

This page lists QoS fields that can be used to send application-specific data during participant and endpoint discovery.

There are also various resource limits related to these QoS which are also listed below.

Basic Settings

DDS_PropertyQosPolicy (p. 1260)	**value** (p. 1262)
	<<***basic***>> *(p. 201)* <<***extension***>> *(p. 213)* Sequence of properties.

Advanced Settings

DDS_UserDataQosPolicy (p. 1504)	**value** (p. 1505)
	<<***spec***>> *(p. 209)* a sequence of octets
DDS_GroupDataQosPolicy (p. 1160)	**value** (p. 1161)
	<<***spec***>> *(p. 209)* a sequence of octets
DDS_TypeSupportQosPolicy (p. 1499)	**plugin_data** (p. 1500)
	[DEPRECATED] (p. 384) <<***extension***>> *(p. 213)* Value to pass into the type plugin's de-/serialization function
DDS_DomainParticipantResourceLimitsQosPolicy (p. 977)	**participant_user_data_max_length** (p. 991)
	<<***extension***>> *(p. 213)* Maximum length of user data in **DDS_DomainParticipantQos** (p. 971) and **DDS_↩ ParticipantBuiltinTopicData** (p. 1229)
DDS_DomainParticipantResourceLimitsQosPolicy (p. 977)	**topic_data_max_length** (p. 991)
	<<***extension***>> *(p. 213)* Maximum length of topic data in **DDS_TopicQos** (p. 1397), **DDS_TopicBuiltin↩ TopicData** (p. 1391), **DDS_PublicationBuiltinTopic↩ Data** (p. 1264) and **DDS_SubscriptionBuiltinTopic↩ Data** (p. 1370)
DDS_DomainParticipantResourceLimitsQosPolicy (p. 977)	**publisher_group_data_max_length** (p. 992)
	<<***extension***>> *(p. 213)* Maximum length of group data in **DDS_PublisherQos** (p. 1276) and **DDS_↩ PublicationBuiltinTopicData** (p. 1264)
DDS_DomainParticipantResourceLimitsQosPolicy (p. 977)	**subscriber_group_data_max_length** (p. 992)
	<<***extension***>> *(p. 213)* Maximum length of group data in **DDS_SubscriberQos** (p. 1367) and **DDS_↩ SubscriptionBuiltinTopicData** (p. 1370)
DDS_DomainParticipantResourceLimitsQosPolicy (p. 977)	**writer_user_data_max_length** (p. 992)
	<<***extension***>> *(p. 213)* Maximum length of user data in **DDS_DataWriterQos** (p. 909) and **DDS_↩ PublicationBuiltinTopicData** (p. 1264)
DDS_DomainParticipantResourceLimitsQosPolicy (p. 977)	**reader_user_data_max_length** (p. 993)
	<<***extension***>> *(p. 213)* Maximum length of user data in **DDS_DataReaderQos** (p. 858) and **DDS_↩ SubscriptionBuiltinTopicData** (p. 1370)

DDS_DomainParticipantResourceLimitsQosPolicy (p. 977)	**participant_property_list_max_length** (p. 998)
	<<***extension***>> *(p. 213)* Maximum number of properties associated with the **DDSDomainParticipant** (p. 1619).
DDS_DomainParticipantResourceLimitsQosPolicy (p. 977)	**participant_property_string_max_length** (p. 998)
	<<***extension***>> *(p. 213)* Maximum string length of the properties associated with the **DDSDomainParticipant** (p. 1619).
DDS_DomainParticipantResourceLimitsQosPolicy (p. 977)	**writer_property_list_max_length** (p. 999)
	<<***extension***>> *(p. 213)* Maximum number of properties associated with a **DDSDataWriter** (p. 1589).
DDS_DomainParticipantResourceLimitsQosPolicy (p. 977)	**writer_property_string_max_length** (p. 999)
	<<***extension***>> *(p. 213)* Maximum string length of the properties associated with a **DDSDataWriter** (p. 1589).
DDS_DomainParticipantResourceLimitsQosPolicy (p. 977)	**reader_property_list_max_length** (p. 999)
	<<***extension***>> *(p. 213)* Maximum number of properties associated with a **DDSDataReader** (p. 1556).
DDS_DomainParticipantResourceLimitsQosPolicy (p. 977)	**reader_property_string_max_length** (p. 1000)
	<<***extension***>> *(p. 213)* Maximum string length of the properties associated with a **DDSDataReader** (p. 1556).

7.53 Application Startup

Use these QoS fields to configure entity creation and application startup settings.

Use these QoS fields to configure entity creation and application startup settings.

Basic Settings

DDS_EntityFactoryQosPolicy (p. 1145)	**autoenable_created_entities** (p. 1147)
	<<***basic***>> *(p. 201)* <<***spec***>> *(p. 209)* Specifies whether the entity acting as a factory automatically enables the instances it creates.

7.54 Batching

Use these QoS fields to configure the batching feature.

Use these QoS fields to configure the batching feature.

See `BATCH QosPolicy (DDS Extension)`, in the *Core Libraries User's Manual*, for more information about this feature.

Basic Settings

DDS_BatchQosPolicy (p. 810)	**enable** (p. 810)
	<<***basic***>> (p. *201)* <<***extension***>> (p. *213)* Specifies whether or not batching is enabled.
DDS_BatchQosPolicy (p. 810)	**max_data_bytes** (p. 811)
	<<***basic***>> (p. *201)* <<***extension***>> (p. *213)* The maximum cumulative length of all serialized samples in a batch.
DDS_BatchQosPolicy (p. 810)	**max_samples** (p. 811)
	<<***basic***>> (p. *201)* <<***extension***>> (p. *213)* The maximum number of samples in a batch.

Advanced Settings

DDS_AsynchronousPublisherQosPolicy (p. 798)	**disable_asynchronous_batch** (p. 800)
	[DEPRECATED] (p. **384)** <<***extension***>> (p. *213)* Disable asynchronous batch flushing.
DDS_AsynchronousPublisherQosPolicy (p. 798)	**asynchronous_batch_thread** (p. 801)
	<<***extension***>> (p. *213)* Settings of the batch flushing thread.
DDS_DataWriterResourceLimitsQosPolicy (p. 917)	**initial_batches** (p. 920)
	<<***extension***>> (p. *213)* Represents the initial number of batches a **DDSDataWriter** (p. 1589) will manage.
DDS_DataWriterResourceLimitsQosPolicy (p. 917)	**max_batches** (p. 921)
	<<***extension***>> (p. *213)* Represents the maximum number of batches a **DDSDataWriter** (p. 1589) will manage.
DDS_BatchQosPolicy (p. 810)	**max_flush_delay** (p. 812)
	<<***extension***>> (p. *213)* The maximum flush delay.
DDS_BatchQosPolicy (p. 810)	**source_timestamp_resolution** (p. 812)

	<<***extension***>> *(p. 213)* Batch source timestamp resolution.
DDS_BatchQosPolicy (p. 810)	**thread_safe_write** (p. 813)
	<<***extension***>> *(p. 213)* Determines whether or not the write operation is thread safe.

7.55 Concurrency

Use these QoS fields to configure threads and concurrency settings in RTI Connext.

Use these QoS fields to configure threads and concurrency settings in RTI Connext.

See `Connext Threading Model`, in the *Core Libraries User's Manual*, for more information.

This community article gives a thorough overview of how and why to use many of the QoS listed below: `https://community.rti.com/kb/achieving-low-jitter-performance-connext-pro`.

Advanced Settings

DDS_TransportPriorityQosPolicy (p. 1419)	**value** (p. 1421)
	<<***spec***>> *(p. 209)* This policy is a hint to the infrastructure as to how to set the priority of the underlying transport used to send the data.
DDS_AsynchronousPublisherQosPolicy (p. 798)	**disable_asynchronous_write** (p. 800)
	[DEPRECATED] (p. **384**) <<***extension***>> *(p. 213)* Disable asynchronous publishing.
DDS_AsynchronousPublisherQosPolicy (p. 798)	**thread** (p. 800)
	<<***extension***>> *(p. 213)* Settings of the publishing thread.
DDS_AsynchronousPublisherQosPolicy (p. 798)	**disable_asynchronous_batch** (p. 800)
	[DEPRECATED] (p. **384**) <<***extension***>> *(p. 213)* Disable asynchronous batch flushing.
DDS_AsynchronousPublisherQosPolicy (p. 798)	**asynchronous_batch_thread** (p. 801)
	<<***extension***>> *(p. 213)* Settings of the batch flushing thread.
DDS_AsynchronousPublisherQosPolicy (p. 798)	**disable_topic_query_publication** (p. 801)
	[DEPRECATED] (p. **384**) <<***extension***>> *(p. 213)* Disable topic query publication.
DDS_AsynchronousPublisherQosPolicy (p. 798)	**topic_query_publication_thread** (p. 802)

	<<***extension***>> *(p. 213)* Settings of the **DDSTopicQuery** (p. 1895) publication thread.
DDS_DiscoveryQosPolicy (p. 960)	**metatraffic_transport_priority** (p. 963)
	<<***extension***>> *(p. 213)* The transport priority to use for the Discovery meta-traffic.
DDS_DataWriterResourceLimitsQosPolicy (p. 917)	**initial_concurrent_blocking_threads** (p. 919)
	[DEPRECATED] (p. **384**) <<***extension***>> *(p. 213)* The initial number of threads that are allowed to concurrently block on write call on the same **DDSDataWriter** (p. 1589).
DDS_DataWriterResourceLimitsQosPolicy (p. 917)	**max_concurrent_blocking_threads** (p. 919)
	[DEPRECATED] (p. **384**) <<***extension***>> *(p. 213)* The maximum number of threads that are allowed to concurrently block on write call on the same **DDSDataWriter** (p. 1589).
DDS_EventQosPolicy (p. 1150)	**thread** (p. 1151)
	<<***extension***>> *(p. 213)* Event thread QoS.
DDS_DatabaseQosPolicy (p. 831)	**thread** (p. 832)
	<<***extension***>> *(p. 213)* Database thread settings.
DDS_ReceiverPoolQosPolicy (p. 1293)	**thread** (p. 1294)
	<<***extension***>> *(p. 213)* Receiver pool thread(s).
DDS_BatchQosPolicy (p. 810)	**thread_safe_write** (p. 813)
	<<***extension***>> *(p. 213)* Determines whether or not the write operation is thread safe.
DDS_MonitoringEventDistributionSettings (p. 1197)	**concurrency_level** (p. 1198)
	<<***extension***>> *(p. 213)* Defines how concurrent the push of event metrics to RTI Monitoring Library 2.0 is.
DDS_MonitoringEventDistributionSettings (p. 1197)	**thread** (p. 1199)
	<<***extension***>> *(p. 213)* The settings of the event metric thread.
DDS_MonitoringPeriodicDistributionSettings (p. 1209)	**thread** (p. 1209)
	<<***extension***>> *(p. 213)* The settings of the periodic metric thread.
DDS_MonitoringLoggingDistributionSettings (p. 1200)	**concurrency_level** (p. 1201)
	<<***extension***>> *(p. 213)* Defines how concurrent the push of log messages to RTI Monitoring Library 2.0 is.

DDS_MonitoringLoggingDistributionSettings (p. 1200)	**thread** (p. 1203)
	<<***extension***>> *(p. 213)* The settings of the logging thread.

7.56 Content Filtering

Use these QoS fields if you are using content filtering on your data.

Use these QoS fields if you are using content filtering on your data.

Content filtering most commonly refers to using a **DDSContentFilteredTopic** (p. 1551), but data can also be filtered using **DDSTopicQuery** (p. 1895) to run a new filter over the data in a DataWriter's cache, or **DDSQueryCondition** (p. 1841) to read a subset of the data that is currently stored in the DataReader's cache.

See also

Queries and Filters Syntax (p. 184).

Advanced Settings

DDS_DataReaderResourceLimitsQosPolicy (p. 867)	**max_query_condition_filters** (p. 880)
	<<***extension***>> *(p. 213)* The maximum number of query condition filters a reader is allowed.
DDS_DataWriterResourceLimitsQosPolicy (p. 917)	**max_remote_reader_filters** (p. 920)
	<<***extension***>> *(p. 213)* The maximum number of remote DataReaders for which the **DDSDataWriter** (p. 1589) will perform content-based filtering.
DDS_DomainParticipantResourceLimitsQosPolicy (p. 977)	**content_filtered_topic_allocation** (p. 984)
	<<***extension***>> *(p. 213)* Allocation settings applied to content filtered topic.
DDS_DomainParticipantResourceLimitsQosPolicy (p. 977)	**content_filter_allocation** (p. 984)
	<<***extension***>> *(p. 213)* Allocation settings applied to content filter.
DDS_DomainParticipantResourceLimitsQosPolicy (p. 977)	**query_condition_allocation** (p. 985)
	<<***extension***>> *(p. 213)* Allocation settings applied to query condition pool.
DDS_DomainParticipantResourceLimitsQosPolicy (p. 977)	**content_filtered_topic_hash_buckets** (p. 990)

	[DEPRECATED] (p. **384**) <<***extension***>> *(p. 213)* Number of hash buckets for content filtered topics.
DDS_DomainParticipantResourceLimitsQosPolicy (p. 977)	**content_filter_hash_buckets** (p. 990)
	[DEPRECATED] (p. **384**) <<***extension***>> *(p. 213)* Number of hash buckets for content filters.
DDS_DomainParticipantResourceLimitsQosPolicy (p. 977)	**contentfilter_property_max_length** (p. 997)
	<<***extension***>> *(p. 213)* This field is the maximum length of all data related to a Content-filtered topic.
DDS_DomainParticipantResourceLimitsQosPolicy (p. 977)	**channel_seq_max_length** (p. 997)
	<<***extension***>> *(p. 213)* Maximum number of channels that can be specified in **DDS_MultiChannelQos↩ Policy** (p. 1214) for MultiChannel DataWriters.
DDS_DomainParticipantResourceLimitsQosPolicy (p. 977)	**channel_filter_expression_max_length** (p. 998)
	<<***extension***>> *(p. 213)* Maximum length of a channel **DDS_ChannelSettings_t::filter_expression** (p. 822) in a MultiChannel DataWriter.
DDS_ChannelSettings_t (p. 822)	**multicast_settings** (p. 822)
	<<***extension***>> *(p. 213)* A sequence of **DDS_↩ TransportMulticastSettings_t** (p. 1417) used to configure the multicast addresses associated with a channel.
DDS_ChannelSettings_t (p. 822)	**filter_expression** (p. 822)
	<<***extension***>> *(p. 213)* A logical expression used to determine the data that will be published in the channel.
DDS_MultiChannelQosPolicy (p. 1214)	**filter_name** (p. 1215)
	<<***extension***>> *(p. 213)* Name of the filter class used to describe the filter expressions of a MultiChannel DataWriter.

7.57 Data Availability

Use these QoS fields to configure how much data, which data, and when that data is available to an application.

Use these QoS fields to configure how much data, which data, and when that data is available to an application.

Data availability refers to configurations such as ownership and lifespan.

Basic Settings

DDS_DurabilityQosPolicy (p. 1005)	**kind** (p. 1007)
	*<<**basic**>> (p. 201) <<**spec**>> (p. 209)* The kind of durability.
DDS_DurabilityQosPolicy (p. 1005)	**writer_depth** (p. 1008)
	*<<**basic**>> (p. 201) <<**extension**>> (p. 213)* Indicates the number of samples per instance that a durable **DDSDataWriter** (p. 1589) will send to a late-joining **DDSDataReader** (p. 1556).
DDS_DeadlineQosPolicy (p. 928)	**period** (p. 930)
	*<<**basic**>> (p. 201) <<**spec**>> (p. 209)* Duration of the deadline period.
DDS_OwnershipQosPolicy (p. 1222)	**kind** (p. 1227)
	*<<**basic**>> (p. 201) <<**spec**>> (p. 209)* The kind of ownership
DDS_OwnershipStrengthQosPolicy (p. 1227)	**value** (p. 1228)
	*<<**basic**>> (p. 201) <<**spec**>> (p. 209)* The strength value used to arbitrate among multiple writers.
DDS_TimeBasedFilterQosPolicy (p. 1389)	**minimum_separation** (p. 1391)
	*<<**basic**>> (p. 201) <<**spec**>> (p. 209)* The minimum separation duration between subsequent samples.
DDS_HistoryQosPolicy (p. 1162)	**kind** (p. 1164)
	*<<**basic**>> (p. 201) <<**spec**>> (p. 209)* Specifies the kind of history to be kept.
DDS_HistoryQosPolicy (p. 1162)	**depth** (p. 1164)
	*<<**basic**>> (p. 201) <<**spec**>> (p. 209)* Specifies the number of samples per instance to be kept, when the `kind` is **DDS_KEEP_LAST_HISTORY_QOS** (p. 604)
DDS_ResourceLimitsQosPolicy (p. 1308)	**max_samples** (p. 1311)
	*<<**basic**>> (p. 201) <<**spec**>> (p. 209)* Represents the maximum samples the middleware can store for any one **DDSDataWriter** (p. 1589) (or **DDSDataReader** (p. 1556)).
DDS_ResourceLimitsQosPolicy (p. 1308)	**max_instances** (p. 1311)
	*<<**basic**>> (p. 201) <<**spec**>> (p. 209)* Represents the maximum number of instances a **DDSDataWriter** (p. 1589) (or **DDSDataReader** (p. 1556)) can manage.
DDS_LifespanQosPolicy (p. 1175)	**duration** (p. 1176)

	<<***basic***>> (p. *201)* <<***spec***>> (p. *209)* Maximum duration for the data's validity.
DDS_MonitoringEventDistributionSettings (p. 1197)	**publication_period** (p. 1200)
	<<***basic***>> (p. *201)* <<***extension***>> (p. *213)* Period at which the event metric thread publishes the event metrics that have changed since the last time they were published.
DDS_MonitoringPeriodicDistributionSettings (p. 1209)	**polling_period** (p. 1210)
	<<***basic***>> (p. *201)* <<***extension***>> (p. *213)* Period at which the periodic metric thread polls and publishes the periodic metrics.
DDS_MonitoringLoggingDistributionSettings (p. 1200)	**publication_period** (p. 1203)
	<<***basic***>> (p. *201)* <<***extension***>> (p. *213)* Period at which the logging thread publishes log messages.

Advanced Settings

DDS_AvailabilityQosPolicy (p. 805)	**enable_required_subscriptions** (p. 808)
	<<***extension***>> (p. *213)* Enables support for required subscriptions in a **DDSDataWriter** (p. 1589)
DDS_AvailabilityQosPolicy (p. 805)	**max_data_availability_waiting_time** (p. 808)
	<<***extension***>> (p. *213)* Defines how much time to wait before delivering a sample to the application without having received some of the previous samples.
DDS_AvailabilityQosPolicy (p. 805)	**max_endpoint_availability_waiting_time** (p. 808)
	<<***extension***>> (p. *213)* Defines how much time to wait to discover DataWriters providing samples for the same data source (virtual GUID).
DDS_AvailabilityQosPolicy (p. 805)	**required_matched_endpoint_groups** (p. 809)
	<<***extension***>> (p. *213)* A sequence of endpoint groups.
DDS_DurabilityQosPolicy (p. 1005)	**direct_communication** (p. 1007)
	<<***extension***>> (p. *213)* Indicates whether or not a TRANSIENT or PERSISTENT **DDSDataReader** (p. 1556) should receive samples directly from a TRANSIENT or PERSISTENT **DDSDataWriter** (p. 1589)
DDS_DurabilityQosPolicy (p. 1005)	**storage_settings** (p. 1008)
	<<***extension***>> (p. *213)* Configures durable writer history and durable reader state.

DDS_DurabilityServiceQosPolicy (p. 1009)	**service_cleanup_delay** (p. 1010)
	<<***spec***>> *(p. 209)* Controls when the service is able to remove all information regarding a data instances.
DDS_DurabilityServiceQosPolicy (p. 1009)	**max_samples** (p. 1011)
	<<***spec***>> *(p. 209)* Part of resource limits QoS policy to apply when feeding a late joiner.
DDS_DurabilityServiceQosPolicy (p. 1009)	**max_instances** (p. 1011)
	<<***spec***>> *(p. 209)* Part of resource limits QoS policy to apply when feeding a late joiner.
DDS_DurabilityServiceQosPolicy (p. 1009)	**max_samples_per_instance** (p. 1012)
	<<***spec***>> *(p. 209)* Part of resource limits QoS policy to apply when feeding a late joiner.
DDS_PresentationQosPolicy (p. 1249)	**access_scope** (p. 1252)
	<<***spec***>> *(p. 209)* Determines the largest scope spanning the entities for which the order and coherency of changes can be preserved.
DDS_PresentationQosPolicy (p. 1249)	**coherent_access** (p. 1252)
	<<***spec***>> *(p. 209)* Specifies support for *coherent* access. Controls whether coherent access is supported within the scope `access_scope`.
DDS_PresentationQosPolicy (p. 1249)	**ordered_access** (p. 1252)
	<<***spec***>> *(p. 209)* Specifies support for *ordered* access to the samples received at the subscription end. Controls whether ordered access is supported within the scope `access_scope`.
DDS_PresentationQosPolicy (p. 1249)	**drop_incomplete_coherent_set** (p. 1253)
	<<***extension***>> *(p. 213)* Indicates whether or not a **DDSDataReader** (p. 1556) should drop samples from an incomplete coherent set (one for which not all the samples were received). Such samples are reported as lost in the SAMPLE_LOST Status.
DDS_LatencyBudgetQosPolicy (p. 1173)	**duration** (p. 1175)
	<<***spec***>> *(p. 209)* Duration of the maximum acceptable delay.
DDS_ResourceLimitsQosPolicy (p. 1308)	**max_samples_per_instance** (p. 1311)
	<<***spec***>> *(p. 209)* Represents the maximum number of samples of any one instance a **DDSDataWriter** (p. 1589) (or **DDSDataReader** (p. 1556)) can manage.
DDS_ReaderDataLifecycleQosPolicy (p. 1289)	**autopurge_nowriter_samples_delay** (p. 1291)

	<<***spec***>> *(p. 209)* Minimum duration for which the **DDSDataReader** (p. 1556) will maintain samples for an instance once its `instance_state` becomes **DDS_NOT_ALIVE_NO_WRITERS_INSTANCE_STATE** (p. 166).
DDS_ReaderDataLifecycleQosPolicy (p. 1289)	**autopurge_disposed_samples_delay** (p. 1291)
	<<***spec***>> *(p. 209)* Minimum duration for which the **DDSDataReader** (p. 1556) will maintain samples for an instance once its `instance_state` becomes **DDS_NOT_ALIVE_DISPOSED_INSTANCE_STATE** (p. 166).
DDS_ReaderDataLifecycleQosPolicy (p. 1289)	**autopurge_disposed_instances_delay** (p. 1291)
	<<***extension***>> *(p. 213)* Minimum duration for which the **DDSDataReader** (p. 1556) will maintain information about a received instance once its `instance_state` becomes **DDS_NOT_ALIVE_DISPOSED_INSTANCE_STATE** (p. 166) and there are no samples for the instance in the DataReader queue.
DDS_ReaderDataLifecycleQosPolicy (p. 1289)	**autopurge_nowriter_instances_delay** (p. 1292)
	<<***extension***>> *(p. 213)* Minimum duration for which the **DDSDataReader** (p. 1556) will maintain information about a received instance once its `instance_state` becomes **DDS_NOT_ALIVE_NO_WRITERS_INSTANCE_STATE** (p. 166) and there are no samples for the instance in the DataReader queue.
DDS_TransportMulticastMappingQosPolicy (p. 1413)	**value** (p. 1415)
	<<***extension***>> *(p. 213)* A sequence of multicast communication mappings.
DDS_TransportMulticastQosPolicy (p. 1415)	**kind** (p. 1416)
	<<***extension***>> *(p. 213)* A value that specifies a way to determine how to obtain the multicast address.
DDS_DataWriterProtocolQosPolicy (p. 892)	**virtual_guid** (p. 893)
	<<***extension***>> *(p. 213)* The virtual GUID (Global Unique Identifier).
DDS_DataWriterProtocolQosPolicy (p. 892)	**push_on_write** (p. 894)
	<<***extension***>> *(p. 213)* Whether to push sample out when write is called.
DDS_BuiltinTopicReaderResourceLimits_t (p. 815)	**initial_outstanding_reads** (p. 817)
	<<***extension***>> *(p. 213)* The initial number of outstanding reads that have not called finish yet on the same built-in topic **DDSDataReader** (p. 1556).

DDS_BuiltinTopicReaderResourceLimits_t (p. 815)	**max_outstanding_reads** (p. 818)
	<<***extension***>> *(p. 213)* The maximum number of outstanding reads that have not called finish yet on the same built-in topic **DDSDataReader** (p. 1556).
DDS_BuiltinTopicReaderResourceLimits_t (p. 815)	**max_samples_per_read** (p. 818)
	<<***extension***>> *(p. 213)* Maximum number of samples that can be read/taken on a same built-in topic **DDSDataReader** (p. 1556).
DDS_BatchQosPolicy (p. 810)	**source_timestamp_resolution** (p. 812)
	<<***extension***>> *(p. 213)* Batch source timestamp resolution.
DDS_ChannelSettings_t (p. 822)	**priority** (p. 823)
	<<***extension***>> *(p. 213)* Publication priority
DDS_MultiChannelQosPolicy (p. 1214)	**channels** (p. 1215)
	<<***extension***>> *(p. 213)* A sequence of **DDS_↩ ChannelSettings_t** (p. 822) used to configure the channels' properties. If the length of the sequence is zero, the QoS policy will be ignored.
DDS_MultiChannelQosPolicy (p. 1214)	**filter_name** (p. 1215)
	<<***extension***>> *(p. 213)* Name of the filter class used to describe the filter expressions of a MultiChannel DataWriter.

7.58 Debuggability

Use these QoS to configure logging and observability capabilities to help debug RTI Connext applications.

Use these QoS to configure logging and observability capabilities to help debug RTI Connext applications.

Basic Settings

DDS_EntityNameQosPolicy (p. 1147)	**name** (p. 1148)
	<<***basic***>> *(p. 201)* <<***extension***>> *(p. 213)* The name of the entity.
DDS_LoggingQosPolicy (p. 1188)	**verbosity** (p. 1189)
	<<***basic***>> *(p. 201)* <<***extension***>> *(p. 213)* The verbosities at which RTI Connext diagnostic information is logged.
DDS_LoggingQosPolicy (p. 1188)	**category** (p. 1189)

	*<<**basic**>> (p. 201) <<**extension**>> (p. 213)* Categories of logged messages.
DDS_LoggingQosPolicy (p. 1188)	**output_file** (p. 1189)
	*<<**basic**>> (p. 201) <<**extension**>> (p. 213)* Specifies the file to which log messages will be redirected to.
DDS_LoggingQosPolicy (p. 1188)	**output_file_suffix** (p. 1190)
	*<<**basic**>> (p. 201) <<**extension**>> (p. 213)* Sets the file suffix when logging to a set of files
DDS_MonitoringEventDistributionSettings (p. 1197)	**publication_period** (p. 1200)
	*<<**basic**>> (p. 201) <<**extension**>> (p. 213)* Period at which the event metric thread publishes the event metrics that have changed since the last time they were published.
DDS_MonitoringPeriodicDistributionSettings (p. 1209)	**polling_period** (p. 1210)
	*<<**basic**>> (p. 201) <<**extension**>> (p. 213)* Period at which the periodic metric thread polls and publishes the periodic metrics.
DDS_MonitoringLoggingDistributionSettings (p. 1200)	**max_historical_logs** (p. 1202)
	*<<**basic**>> (p. 201) <<**extension**>> (p. 213)* The number of log messages that RTI Monitoring Library 2.0 will keep as history.
DDS_MonitoringLoggingDistributionSettings (p. 1200)	**publication_period** (p. 1203)
	*<<**basic**>> (p. 201) <<**extension**>> (p. 213)* Period at which the logging thread publishes log messages.
DDS_MonitoringDedicatedParticipantSettings (p. 1193)	**enable** (p. 1193)
	*<<**basic**>> (p. 201) <<**extension**>> (p. 213)* Enables the use of a dedicated **DDSDomainParticipant** (p. 1619) to distribute the RTI Connext application telemetry data.
DDS_MonitoringDedicatedParticipantSettings (p. 1193)	**domain_id** (p. 1194)
	*<<**basic**>> (p. 201) <<**extension**>> (p. 213)* The domain ID used in the creation of RTI Monitoring Library 2.0 **DDSDomainParticipant** (p. 1619).
DDS_MonitoringDedicatedParticipantSettings (p. 1193)	**collector_initial_peers** (p. 1194)

	*<<**basic**>> (p. 201) <<**extension**>> (p. 213)* Determines the initial list of peers that the discovery process will contact to send announcements about the presence of the **DDS_MonitoringDistributionSettings::dedicated_participant** (p. 1196).
DDS_MonitoringLoggingForwardingSettings (p. 1204)	**middleware_forwarding_level** (p. 1204)
	*<<**basic**>> (p. 201) <<**extension**>> (p. 213)* Log messages with **NDDS_CONFIG_LOG_FACILITY_MIDDLEWARE** (p. 727) and a log level more verbose than this value will not be distributed by RTI Monitoring Library 2.0.
DDS_MonitoringLoggingForwardingSettings (p. 1204)	**security_event_forwarding_level** (p. 1204)
	*<<**basic**>> (p. 201) <<**extension**>> (p. 213)* Log messages with **NDDS_CONFIG_LOG_FACILITY_SECURITY_EVENT** (p. 727) and a log level more verbose than this value will not be distributed by RTI Monitoring Library 2.0.
DDS_MonitoringLoggingForwardingSettings (p. 1204)	**service_forwarding_level** (p. 1205)
	*<<**basic**>> (p. 201) <<**extension**>> (p. 213)* Log messages with **NDDS_CONFIG_LOG_FACILITY_SERVICE** (p. 727) and a log level more verbose than this value will not be distributed by RTI Monitoring Library 2.0.
DDS_MonitoringLoggingForwardingSettings (p. 1204)	**user_forwarding_level** (p. 1205)
	*<<**basic**>> (p. 201) <<**extension**>> (p. 213)* Log messages with **NDDS_CONFIG_LOG_FACILITY_USER** (p. 727) and a log level more verbose than this value will not be distributed by RTI Monitoring Library 2.0.
DDS_MonitoringTelemetryData (p. 1213)	**metrics** (p. 1213)
	*<<**basic**>> (p. 201) <<**extension**>> (p. 213)* Sequence of **DDS_MonitoringMetricSelection** (p. 1206) containing the event and periodic metrics that will be collected and distributed for a given set of observable resources.
DDS_MonitoringTelemetryData (p. 1213)	**logs** (p. 1213)
	*<<**basic**>> (p. 201) <<**extension**>> (p. 213)* **DDS_MonitoringLoggingForwardingSettings** (p. 1204) containing the **NDDS_Config_Syslog_Verbosity** (p. 727) levels that will be forwarded for the different **NDDS_Config_LogFacility** (p. 727).

DDS_MonitoringQosPolicy (p. 1210)	**enable** (p. 1211)
	<<***basic***>> *(p. 201)* <<***extension***>> *(p. 213)* Enables the collection and distribution of telemetry data for an RTI Connext application using RTI Monitoring Library 2.0.
DDS_MonitoringQosPolicy (p. 1210)	**application_name** (p. 1211)
	<<***basic***>> *(p. 201)* <<***extension***>> *(p. 213)* The name of the resource that represents this RTI Connext application.

Advanced Settings

DDS_DiscoveryConfigQosPolicy (p. 934)	**default_domain_announcement_period** (p. 949)
	<<***extension***>> *(p. 213)* The period to announce a participant to the default domain 0.
DDS_DiscoveryConfigQosPolicy (p. 934)	**ignore_default_domain_announcements** (p. 950)
	[DEPRECATED] (p. **384)** <<***extension***>> *(p. 213)* Used to ignore the announcements received by a participant on the default domain 0 corresponding to participants running on domains IDs other than 0.
DDS_EntityNameQosPolicy (p. 1147)	**role_name** (p. 1148)
	<<***extension***>> *(p. 213)* The entity role name.
DDS_ServiceQosPolicy (p. 1358)	**kind** (p. 1359)
	<<***extension***>> *(p. 213)* The kind of service.
DDS_LoggingQosPolicy (p. 1188)	**print_format** (p. 1189)
	<<***extension***>> *(p. 213)* The format used to output RTI Connext diagnostic information.
DDS_LoggingQosPolicy (p. 1188)	**max_bytes_per_file** (p. 1190)
	<<***extension***>> *(p. 213)* Specifies the maximum number of bytes a single file can contain
DDS_LoggingQosPolicy (p. 1188)	**max_files** (p. 1191)
	<<***extension***>> *(p. 213)* Specifies the maximum number of files to create before overwritting the previous ones.
DDS_MonitoringEventDistributionSettings (p. 1197)	**concurrency_level** (p. 1198)
	<<***extension***>> *(p. 213)* Defines how concurrent the push of event metrics to RTI Monitoring Library 2.0 is.
DDS_MonitoringEventDistributionSettings (p. 1197)	**datawriter_qos_profile_name** (p. 1199)

	<<***extension***>> *(p. 213)* The fully qualified name of the profile used to configure the **DDSDataWriter** (p. 1589) that distributes event metrics.
DDS_MonitoringEventDistributionSettings (p. 1197)	**thread** (p. 1199)
	<<***extension***>> *(p. 213)* The settings of the event metric thread.
DDS_MonitoringPeriodicDistributionSettings (p. 1209)	**datawriter_qos_profile_name** (p. 1209)
	<<***extension***>> *(p. 213)* The fully qualified name of the profile used to configure the **DDSDataWriter** (p. 1589) that distributes periodic metrics.
DDS_MonitoringPeriodicDistributionSettings (p. 1209)	**thread** (p. 1209)
	<<***extension***>> *(p. 213)* The settings of the periodic metric thread.
DDS_MonitoringLoggingDistributionSettings (p. 1200)	**concurrency_level** (p. 1201)
	<<***extension***>> *(p. 213)* Defines how concurrent the push of log messages to RTI Monitoring Library 2.0 is.
DDS_MonitoringLoggingDistributionSettings (p. 1200)	**datawriter_qos_profile_name** (p. 1202)
	<<***extension***>> *(p. 213)* The fully qualified name of the profile used to configure the **DDSDataWriter** (p. 1589) that distributes log messages.
DDS_MonitoringLoggingDistributionSettings (p. 1200)	**thread** (p. 1203)
	<<***extension***>> *(p. 213)* The settings of the logging thread.
DDS_MonitoringDedicatedParticipantSettings (p. 1193)	**participant_qos_profile_name** (p. 1194)
	<<***extension***>> *(p. 213)* The fully qualified name of the profile used to configure the **DDSDomain↩ Participant** (p. 1619) that will be used to distribute telemetry data.
DDS_MonitoringDistributionSettings (p. 1195)	**dedicated_participant** (p. 1196)
	<<***extension***>> *(p. 213)* Configures the use of a dedicated **DDSDomainParticipant** (p. 1619) to distribute the RTI Connext application telemetry data.
DDS_MonitoringDistributionSettings (p. 1195)	**publisher_qos_profile_name** (p. 1196)
	<<***extension***>> *(p. 213)* The fully qualified name of the profile used to configure the Publishers that distribute telemetry data.

DDS_MonitoringDistributionSettings (p. 1195)	**event_settings** (p. 1196)
	<<***extension***>> *(p. 213)* Configures the distribution of event metrics.
DDS_MonitoringDistributionSettings (p. 1195)	**periodic_settings** (p. 1197)
	<<***extension***>> *(p. 213)* Configures the distribution of periodic metrics.
DDS_MonitoringDistributionSettings (p. 1195)	**logging_settings** (p. 1197)
	<<***extension***>> *(p. 213)* Configures the distribution of logging messages.
DDS_MonitoringMetricSelection (p. 1206)	**resource_selection** (p. 1206)
	<<***extension***>> *(p. 213)* An expression pattern that selects a subset of resources by matching the pattern to the resource names. **DDS_MonitoringMetric↩ Selection::enabled_metrics_selection** (p. 1207) and **DDS_MonitoringMetricSelection::disabled_↩ metrics_selection** (p. 1207) are applied to this subset of resources.
DDS_MonitoringMetricSelection (p. 1206)	**enabled_metrics_selection** (p. 1207)
	<<***extension***>> *(p. 213)* A sequence of POSIX fnmatch patterns that match the names of the metrics that should be collected and distributed for the observable resources selected by **DDS_MonitoringMetric↩ Selection::resource_selection** (p. 1206).
DDS_MonitoringMetricSelection (p. 1206)	**disabled_metrics_selection** (p. 1207)
	<<***extension***>> *(p. 213)* A sequence of POSIX fnmatch patterns that mach the names of the metrics that should not be collected and distributed for the observable resources selected by **DDS_MonitoringMetric↩ Selection::resource_selection** (p. 1206).
DDS_MonitoringQosPolicy (p. 1210)	**distribution_settings** (p. 1212)
	<<***extension***>> *(p. 213)* Configures the distribution of telemetry data.
DDS_MonitoringQosPolicy (p. 1210)	**telemetry_data** (p. 1212)
	<<***extension***>> *(p. 213)* Configures the telemetry data that will be distributed.

7.59 Discovery

Use these QoS fields to configure how applications discover each other.

Use these QoS fields to configure how applications discover each other.

See `Discovery Overview`, in the *Core Libraries User's Manual*, for more information about discovery.

Basic Settings

DDS_WireProtocolQosPolicy (p. 1511)	**participant_id** (p. 1514)
	<<***basic***>> *(p. 201)* <<***extension***>> *(p. 213)* A value used to distinguish among different participants belonging to the same domain on the same host.
DDS_DiscoveryQosPolicy (p. 960)	**initial_peers** (p. 962)
	<<***basic***>> *(p. 201)* <<***extension***>> *(p. 213)* Determines the initial list of peers that will be contacted by the Discovery mechanism to send announcements about the presence of this participant.
DDS_DiscoveryQosPolicy (p. 960)	**multicast_receive_addresses** (p. 962)
	<<***basic***>> *(p. 201)* <<***extension***>> *(p. 213)* Specifies the multicast group addresses on which discovery-related **meta-traffic** can be received by the Domain↩ Participant.
DDS_DiscoveryQosPolicy (p. 960)	**accept_unknown_peers** (p. 963)
	<<***basic***>> *(p. 201)* <<***extension***>> *(p. 213)* Whether to accept a new participant that is not in the initial peers list.
DDS_DiscoveryConfigQosPolicy (p. 934)	**participant_liveliness_lease_duration** (p. 937)
	<<***basic***>> *(p. 201)* <<***extension***>> *(p. 213)* The liveliness lease duration for the participant.
DDS_DiscoveryConfigQosPolicy (p. 934)	**participant_liveliness_assert_period** (p. 937)
	<<***basic***>> *(p. 201)* <<***extension***>> *(p. 213)* The period to assert liveliness for the participant.
DDS_DiscoveryConfigQosPolicy (p. 934)	**participant_announcement_period** (p. 937)
	<<***basic***>> *(p. 201)* <<***extension***>> *(p. 213)* The period at which a participant announces itself to potential peers when using the Simple Participant Discovery Protocol 2.0 (SPDP2).
DDS_DiscoveryConfigQosPolicy (p. 934)	**initial_participant_announcements** (p. 939)
	<<***basic***>> *(p. 201)* <<***extension***>> *(p. 213)* The number of initial announcements sent when a participant is first enabled.
DDS_DiscoveryConfigQosPolicy (p. 934)	**new_remote_participant_announcements** (p. 939)
	<<***basic***>> *(p. 201)* <<***extension***>> *(p. 213)* The number of participant announcements sent when a remote participant is newly discovered.

DDS_DiscoveryConfigQosPolicy (p. 934)	**min_initial_participant_announcement_period** (p. 940)
	<<***basic***>> *(p. 201)* <<***extension***>> *(p. 213)* The minimum period between initial announcements when a participant is first enabled or when a remote participant is newly discovered.
DDS_DiscoveryConfigQosPolicy (p. 934)	**max_initial_participant_announcement_period** (p. 940)
	<<***basic***>> *(p. 201)* <<***extension***>> *(p. 213)* The maximum period between initial announcements when a participant is first enabled or when a remote participant is newly discovered.
DDS_MonitoringDedicatedParticipantSettings (p. 1193)	**domain_id** (p. 1194)
	<<***basic***>> *(p. 201)* <<***extension***>> *(p. 213)* The domain ID used in the creation of RTI Monitoring Library 2.0 **DDSDomainParticipant** (p. 1619).
DDS_MonitoringDedicatedParticipantSettings (p. 1193)	**collector_initial_peers** (p. 1194)
	<<***basic***>> *(p. 201)* <<***extension***>> *(p. 213)* Determines the initial list of peers that the discovery process will contact to send announcements about the presence of the **DDS_MonitoringDistributionSettings::dedicated_participant** (p. 1196).

Advanced Settings

DDS_WireProtocolQosPolicy (p. 1511)	**rtps_host_id** (p. 1515)
	<<***extension***>> *(p. 213)* The RTPS Host ID of the domain participant.
DDS_WireProtocolQosPolicy (p. 1511)	**rtps_app_id** (p. 1516)
	<<***extension***>> *(p. 213)* The RTPS App ID of the domain participant.
DDS_WireProtocolQosPolicy (p. 1511)	**rtps_instance_id** (p. 1516)
	<<***extension***>> *(p. 213)* The RTPS Instance ID of the **DDSDomainParticipant** (p. 1619).
DDS_WireProtocolQosPolicy (p. 1511)	**rtps_well_known_ports** (p. 1517)
	<<***extension***>> *(p. 213)* Configures the RTPS well-known port mappings.
DDS_WireProtocolQosPolicy (p. 1511)	**rtps_reserved_port_mask** (p. 1517)

	<<***extension***>> *(p. 213)* Specifies which well-known ports to reserve when enabling the participant.
DDS_WireProtocolQosPolicy (p. 1511)	**rtps_auto_id_kind** (p. 1518)
	<<***extension***>> *(p. 213)* Kind of auto mechanism used to calculate the GUID prefix.
DDS_DiscoveryQosPolicy (p. 960)	**enabled_transports** (p. 961)
	<<***extension***>> *(p. 213)* The transports available for use by the Discovery mechanism.
DDS_DiscoveryQosPolicy (p. 960)	**metatraffic_transport_priority** (p. 963)
	<<***extension***>> *(p. 213)* The transport priority to use for the Discovery meta-traffic.
DDS_DiscoveryQosPolicy (p. 960)	**enable_endpoint_discovery** (p. 964)
	[DEPRECATED] (p. 384) <<***extension***>> *(p. 213)* Whether to automatically enable endpoint discovery for all the remote participants.
DDS_DiscoveryConfigQosPolicy (p. 934)	**remote_participant_purge_kind** (p. 938)
	[DEPRECATED] (p. 384) <<***extension***>> *(p. 213)* The participant's behavior for maintaining knowledge of remote participants (and their contained entities) with which discovery communication has been lost.
DDS_DiscoveryConfigQosPolicy (p. 934)	**max_liveliness_loss_detection_period** (p. 938)
	[DEPRECATED] (p. 384) <<***extension***>> *(p. 213)* The maximum amount of time between when a remote entity stops maintaining its liveliness and when the matched local entity realizes that fact.
DDS_DiscoveryConfigQosPolicy (p. 934)	**participant_reader_resource_limits** (p. 941)
	<<***extension***>> *(p. 213)* Resource limits.
DDS_DiscoveryConfigQosPolicy (p. 934)	**publication_reader** (p. 941)
	<<***extension***>> *(p. 213)* RTPS protocol-related configuration settings for the RTPS reliable reader associated with a built-in publication reader.
DDS_DiscoveryConfigQosPolicy (p. 934)	**publication_reader_resource_limits** (p. 942)
	<<***extension***>> *(p. 213)* Resource limits.
DDS_DiscoveryConfigQosPolicy (p. 934)	**subscription_reader** (p. 942)
	<<***extension***>> *(p. 213)* RTPS protocol-related configuration settings for the RTPS reliable reader associated with a built-in subscription reader.
DDS_DiscoveryConfigQosPolicy (p. 934)	**subscription_reader_resource_limits** (p. 943)
	<<***extension***>> *(p. 213)* Resource limits.
DDS_DiscoveryConfigQosPolicy (p. 934)	**participant_message_writer** (p. 947)

	<<***extension***>> *(p. 213)* RTPS protocol-related configuration settings for the RTPS reliable writer associated with the built-in participant message writer. This parameter only has effect if the matching participant message reader is configured with **DDS_RELIABLE_RELIABILITY_QOS** (p. 633) **DDS_Reliability↩ QosPolicyKind** (p. 632).
DDS_DiscoveryConfigQosPolicy (p. 934)	**publication_writer_publish_mode** (p. 948)
	<<***extension***>> *(p. 213)* Publish mode policy for the built-in publication writer.
DDS_DiscoveryConfigQosPolicy (p. 934)	**subscription_writer_publish_mode** (p. 949)
	<<***extension***>> *(p. 213)* Publish mode policy for the built-in subscription writer.
DDS_DiscoveryConfigQosPolicy (p. 934)	**asynchronous_publisher** (p. 949)
	<<***extension***>> *(p. 213)* Asynchronous publishing settings for the discovery **DDSPublisher** (p. 1818) and all entities that are created by it.
DDS_DiscoveryConfigQosPolicy (p. 934)	**default_domain_announcement_period** (p. 949)
	<<***extension***>> *(p. 213)* The period to announce a participant to the default domain 0.
DDS_DiscoveryConfigQosPolicy (p. 934)	**ignore_default_domain_announcements** (p. 950)
	[DEPRECATED] (p. 384) <<***extension***>> *(p. 213)* Used to ignore the announcements received by a participant on the default domain 0 corresponding to participants running on domains IDs other than 0.
DDS_DiscoveryConfigQosPolicy (p. 934)	**participant_message_reader_reliability_kind** (p. 946)
	[DEPRECATED] (p. 384) <<***extension***>> *(p. 213)* Reliability policy for a built-in participant message reader.
DDS_DiscoveryConfigQosPolicy (p. 934)	**participant_message_reader** (p. 947)
	<<***extension***>> *(p. 213)* RTPS reliable reader protocol-related configuration settings for a built-in participant message reader. This parameter only has effect if **DDS_DiscoveryConfigQosPolicy↩ ::participant_message_reader_reliability_kind** (p. 946) is set to **DDS_RELIABLE_RELIABILITY_QOS** (p. 633).
DDS_DiscoveryConfigQosPolicy (p. 934)	**publication_writer** (p. 943)
	<<***extension***>> *(p. 213)* RTPS protocol-related configuration settings for the RTPS reliable writer associated with a built-in publication writer.
DDS_DiscoveryConfigQosPolicy (p. 934)	**publication_writer_data_lifecycle** (p. 944)
	<<***extension***>> *(p. 213)* Writer data lifecycle settings for a built-in publication writer.
DDS_DiscoveryConfigQosPolicy (p. 934)	**subscription_writer** (p. 945)

	<<***extension***>> *(p. 213)* RTPS protocol-related configuration settings for the RTPS reliable writer associated with a built-in subscription writer.
DDS_DiscoveryConfigQosPolicy (p. 934)	**subscription_writer_data_lifecycle** (p. 945)
	<<***extension***>> *(p. 213)* Writer data lifecycle settings for a built-in subscription writer.
DDS_DiscoveryConfigQosPolicy (p. 934)	**builtin_discovery_plugins** (p. 946)
	<<***extension***>> *(p. 213)* Mask of built-in discovery plugin kinds.
DDS_DiscoveryConfigQosPolicy (p. 934)	**service_request_writer** (p. 950)
	<<***extension***>> *(p. 213)* RTPS protocol-related configuration settings for the RTPS reliable writer associated with the built-in **DDS_↩ ServiceRequest** (p. 1359) writer.
DDS_DiscoveryConfigQosPolicy (p. 934)	**service_request_writer_data_lifecycle** (p. 951)
	<<***extension***>> *(p. 213)* Writer data lifecycle settings for a built-in **DDS_ServiceRequest** (p. 1359) writer.
DDS_DiscoveryConfigQosPolicy (p. 934)	**service_request_writer_publish_mode** (p. 951)
	<<***extension***>> *(p. 213)* Publish mode policy for the built-in service request writer.
DDS_DiscoveryConfigQosPolicy (p. 934)	**service_request_reader** (p. 952)
	<<***extension***>> *(p. 213)* RTPS reliable reader protocol-related configuration settings for a built-in **DDS_ServiceRequest** (p. 1359) reader.
DDS_DiscoveryConfigQosPolicy (p. 934)	**secure_volatile_writer** (p. 954)
	<<***extension***>> *(p. 213)* RTPS protocol-related configuration settings for the RTPS reliable writer associated with the built-in secure volatile writer.
DDS_DiscoveryConfigQosPolicy (p. 934)	**secure_volatile_writer_publish_mode** (p. 955)
	<<***extension***>> *(p. 213)* Publish mode policy for the built-in secure volatile writer.
DDS_DiscoveryConfigQosPolicy (p. 934)	**secure_volatile_reader** (p. 955)
	<<***extension***>> *(p. 213)* RTPS reliable reader protocol-related configuration settings for the built-in secure volatile reader.
DDS_DiscoveryConfigQosPolicy (p. 934)	**endpoint_type_object_lb_serialization_threshold** (p. 956)
	<<***extension***>> *(p. 213)* Controls when/if the TypeObject v1 is compressed when propagated in Simple Endpoint Discovery.
DDS_DiscoveryConfigQosPolicy (p. 934)	**dns_tracker_polling_period** (p. 956)
	<<***extension***>> *(p. 213)* Duration that specifies the period used by the DNS tracker to poll the DNS service and check for changes in the hostnames.

DDS_DiscoveryConfigQosPolicy (p. 934)	**participant_configuration_reader** (p. 959)
	<<***extension***>> *(p. 213)* RTPS protocol-related configuration settings for the RTPS reliable reader associated with a built-in participant configuration reader.
DDS_DiscoveryConfigQosPolicy (p. 934)	**participant_configuration_reader_resource_limits** (p. 959)
	<<***extension***>> *(p. 213)* Resource limits for the built-in topic participant configuration reader.
DDS_DiscoveryConfigQosPolicy (p. 934)	**participant_configuration_writer_publish_mode** (p. 957)
	<<***extension***>> *(p. 213)* Publish mode policy for the built-in participant configuration writer.
DDS_DiscoveryConfigQosPolicy (p. 934)	**participant_configuration_writer** (p. 957)
	<<***extension***>> *(p. 213)* RTPS protocol-related configuration settings for the RTPS reliable writer associated with a built-in participant configuration writer.
DDS_DiscoveryConfigQosPolicy (p. 934)	**participant_configuration_writer_data_lifecycle** (p. 958)
	<<***extension***>> *(p. 213)* Writer data lifecycle settings for a built-in participant configuration writer.
DDS_DiscoveryConfigQosPolicy (p. 934)	**request_types_filter** (p. 959)
	<<***extension***>> *(p. 213)* Topic name pattern for requesting all types via Type Lookup Service.

7.60 Discovery Isolation

Use these QoS fields to isolate applications from each other.

Use these QoS fields to isolate applications from each other.

Applications can be isolated through mechanisms such as domain ids, domain tags, and partitions.

See `Discovery Overview`, in the *Core Libraries User's Manual*, for more information about discovery.

Basic Settings

DDS_PartitionQosPolicy (p. 1240)	**name** (p. 1242)
	<<***basic***>> *(p. 201)* <<***spec***>> *(p. 209)* A list of partition names
DDS_DiscoveryQosPolicy (p. 960)	**initial_peers** (p. 962)
	<<***basic***>> *(p. 201)* <<***extension***>> *(p. 213)* Determines the initial list of peers that will be contacted by the Discovery mechanism to send announcements about the presence of this participant.

DDS_DiscoveryQosPolicy (p. 960)	**accept_unknown_peers** (p. 963)
	<<***basic***>> *(p. 201)* <<***extension***>> *(p. 213)* Whether to accept a new participant that is not in the initial peers list.
DDS_MonitoringDedicatedParticipantSettings (p. 1193)	**domain_id** (p. 1194)
	<<***basic***>> *(p. 201)* <<***extension***>> *(p. 213)* The domain ID used in the creation of RTI Monitoring Library 2.0 **DDSDomainParticipant** (p. 1619).
DDS_MonitoringDedicatedParticipantSettings (p. 1193)	**collector_initial_peers** (p. 1194)
	<<***basic***>> *(p. 201)* <<***extension***>> *(p. 213)* Determines the initial list of peers that the discovery process will contact to send announcements about the presence of the **DDS_MonitoringDistributionSettings**↩ **::dedicated_participant** (p. 1196).

Advanced Settings

DDS_DiscoveryQosPolicy (p. 960)	**enable_endpoint_discovery** (p. 964)
	[DEPRECATED] (p. **384**) <<***extension***>> *(p. 213)* Whether to automatically enable endpoint discovery for all the remote participants.
DDS_DomainParticipantResourceLimitsQosPolicy (p. 977)	**max_partitions** (p. 993)
	<<***extension***>> *(p. 213)* Maximum number of partition name strings allowable in a **DDS_PartitionQosPolicy** (p. 1240).
DDS_DomainParticipantResourceLimitsQosPolicy (p. 977)	**max_partition_cumulative_characters** (p. 993)
	<<***extension***>> *(p. 213)* Maximum number of combined characters allowable in all partition names in a **DDS_PartitionQosPolicy** (p. 1240).

7.61 Durability

Use these QoS fields to configure how much and how long data remains retrievable in the system.

Use these QoS fields to configure how much and how long data remains retrievable in the system.

Basic Settings

DDS_DurabilityQosPolicy (p. 1005)	**kind** (p. 1007)
	*<<**basic**>> (p. 201) <<**spec**>> (p. 209)* The kind of durability.
DDS_DurabilityQosPolicy (p. 1005)	**writer_depth** (p. 1008)
	*<<**basic**>> (p. 201) <<**extension**>> (p. 213)* Indicates the number of samples per instance that a durable **DDSDataWriter** (p. 1589) will send to a late-joining **DDSDataReader** (p. 1556).
DDS_ResourceLimitsQosPolicy (p. 1308)	**max_samples** (p. 1311)
	*<<**basic**>> (p. 201) <<**spec**>> (p. 209)* Represents the maximum samples the middleware can store for any one **DDSDataWriter** (p. 1589) (or **DDSDataReader** (p. 1556)).
DDS_ResourceLimitsQosPolicy (p. 1308)	**max_instances** (p. 1311)
	*<<**basic**>> (p. 201) <<**spec**>> (p. 209)* Represents the maximum number of instances a **DDSDataWriter** (p. 1589) (or **DDSData↩ Reader** (p. 1556)) can manage.
DDS_LifespanQosPolicy (p. 1175)	**duration** (p. 1176)
	*<<**basic**>> (p. 201) <<**spec**>> (p. 209)* Maximum duration for the data's validity.

Advanced Settings

DDS_AvailabilityQosPolicy (p. 805)	**enable_required_subscriptions** (p. 808)
	*<<**extension**>> (p. 213)* Enables support for required subscriptions in a **DDSDataWriter** (p. 1589)
DDS_AvailabilityQosPolicy (p. 805)	**max_data_availability_waiting_time** (p. 808)
	*<<**extension**>> (p. 213)* Defines how much time to wait before delivering a sample to the application without having received some of the previous samples.
DDS_AvailabilityQosPolicy (p. 805)	**max_endpoint_availability_waiting_time** (p. 808)
	*<<**extension**>> (p. 213)* Defines how much time to wait to discover DataWriters providing samples for the same data source (virtual GUID).
DDS_AvailabilityQosPolicy (p. 805)	**required_matched_endpoint_groups** (p. 809)
	*<<**extension**>> (p. 213)* A sequence of endpoint groups.
DDS_DurabilityQosPolicy (p. 1005)	**direct_communication** (p. 1007)
	*<<**extension**>> (p. 213)* Indicates whether or not a TRANSIENT or PERSISTENT **DDSDataReader** (p. 1556) should receive samples directly from a TRANSIENT or PERSISTENT **DDSDataWriter** (p. 1589)

DDS_DurabilityQosPolicy (p. 1005)	**storage_settings** (p. 1008)
	<<***extension***>> *(p. 213)* Configures durable writer history and durable reader state.
DDS_PersistentStorageSettings (p. 1242)	**enable** (p. 1244)
	<<***extension***>> *(p. 213)* Enables durable writer history in a **DDSDataWriter** (p. 1589) and durable reader state in a **DDSDataReader** (p. 1556).
DDS_PersistentStorageSettings (p. 1242)	**file_name** (p. 1244)
	<<***extension***>> *(p. 213)* The file name where the durable writer history or durable reader state will be stored.
DDS_PersistentStorageSettings (p. 1242)	**trace_file_name** (p. 1244)
	<<***extension***>> *(p. 213)* The file name where to store the SQL statements executed when loading and storing the durable writer history or durable reader state.
DDS_PersistentStorageSettings (p. 1242)	**journal_kind** (p. 1245)
	<<***extension***>> *(p. 213)* Sets the journal mode of the persistent storage.
DDS_PersistentStorageSettings (p. 1242)	**synchronization_kind** (p. 1245)
	<<***extension***>> *(p. 213)* Sets the level of synchronization with the physical disk.
DDS_PersistentStorageSettings (p. 1242)	**vacuum** (p. 1245)
	<<***extension***>> *(p. 213)* Sets the auto-vacuum status of the storage.
DDS_PersistentStorageSettings (p. 1242)	**restore** (p. 1246)
	<<***extension***>> *(p. 213)* Indicates if the persisted writer history or reader state must be restored.
DDS_PersistentStorageSettings (p. 1242)	**writer_instance_cache_allocation** (p. 1246)
	<<***extension***>> *(p. 213)* Configures the resource limits associated with the instance durable writer history cache.
DDS_PersistentStorageSettings (p. 1242)	**writer_sample_cache_allocation** (p. 1247)
	<<***extension***>> *(p. 213)* Configures the resource limits associated with the sample durable writer history cache.
DDS_PersistentStorageSettings (p. 1242)	**writer_memory_state** (p. 1248)
	<<***extension***>> *(p. 213)* Determines how much state will be kept in memory by the durable writer history in order to avoid accessing the persistent storage in disk.

DDS_PersistentStorageSettings (p. 1242)	**reader_checkpoint_frequency** (p. 1248)
	<<***extension***>> *(p. 213)* Controls how often the reader state is stored into the database.
DDS_DurabilityServiceQosPolicy (p. 1009)	**service_cleanup_delay** (p. 1010)
	<<***spec***>> *(p. 209)* Controls when the service is able to remove all information regarding a data instances.
DDS_DurabilityServiceQosPolicy (p. 1009)	**history_kind** (p. 1010)
	<<***spec***>> *(p. 209)* The kind of history to apply in recouping durable data.
DDS_DurabilityServiceQosPolicy (p. 1009)	**history_depth** (p. 1011)
	<<***spec***>> *(p. 209)* Setting to use for the **DDS_DurabilityQosPolicy::writer_depth** (p. 1008) when recouping durable data.
DDS_DurabilityServiceQosPolicy (p. 1009)	**max_samples** (p. 1011)
	<<***spec***>> *(p. 209)* Part of resource limits QoS policy to apply when feeding a late joiner.
DDS_DurabilityServiceQosPolicy (p. 1009)	**max_instances** (p. 1011)
	<<***spec***>> *(p. 209)* Part of resource limits QoS policy to apply when feeding a late joiner.
DDS_DurabilityServiceQosPolicy (p. 1009)	**max_samples_per_instance** (p. 1012)
	<<***spec***>> *(p. 209)* Part of resource limits QoS policy to apply when feeding a late joiner.
DDS_ResourceLimitsQosPolicy (p. 1308)	**max_samples_per_instance** (p. 1311)
	<<***spec***>> *(p. 209)* Represents the maximum number of samples of any one instance a **DDSDataWriter** (p. 1589) (or **DDSDataReader** (p. 1556)) can manage.
DDS_DataWriterProtocolQosPolicy (p. 892)	**propagate_app_ack_with_no_response** (p. 896)
	<<***extension***>> *(p. 213)* Controls whether or not a **DDSDataWriter** (p. 1589) receives **DDSDataWriterListener::on_application_acknowledgment** (p. 1617) notifications with an empty or invalid response.
DDS_RtpsReliableWriterProtocol_t (p. 1320)	**disable_positive_acks_enable_adaptive_sample_keep_duration** (p. 1331)
	[DEPRECATED] (p. 384) <<***extension***>> *(p. 213)* Enables dynamic adjustment of sample keep duration in response to congestion.
DDS_RtpsReliableWriterProtocol_t (p. 1320)	**disable_positive_acks_min_sample_keep_duration** (p. 1330)

	<<***extension***>> *(p. 213)* The minimum duration a sample is queued for ACK-disabled readers.
DDS_RtpsReliableWriterProtocol_t (p. 1320)	**disable_positive_acks_max_sample_keep_duration** (p. 1331)
	<<***extension***>> *(p. 213)* The maximum duration a sample is queued for ACK-disabled readers.
DDS_RtpsReliableWriterProtocol_t (p. 1320)	**disable_positive_acks_decrease_sample_keep_↩ duration_factor** (p. 1332)
	[DEPRECATED] (p. **384)** <<***extension***>> *(p. 213)* Controls rate of contraction of dynamic sample keep duration.
DDS_RtpsReliableWriterProtocol_t (p. 1320)	**disable_positive_acks_increase_sample_keep_↩ duration_factor** (p. 1332)
	[DEPRECATED] (p. **384)** <<***extension***>> *(p. 213)* Controls rate of growth of dynamic sample keep duration.
DDS_BuiltinTopicReaderResourceLimits_t (p. 815)	**initial_samples** (p. 816)
	<<***extension***>> *(p. 213)* Initial number of samples.
DDS_BuiltinTopicReaderResourceLimits_t (p. 815)	**max_samples** (p. 816)
	<<***extension***>> *(p. 213)* Maximum number of samples.
DDS_EntityNameQosPolicy (p. 1147)	**role_name** (p. 1148)
	<<***extension***>> *(p. 213)* The entity role name.

7.62 Flow Control

Use these QoS fields to configure features which help to control the rate at which data is sent, preventing network congestion.

Use these QoS fields to configure features which help to control the rate at which data is sent, preventing network congestion.

Basic Settings

DDS_ReliabilityQosPolicy (p. 1295)	**max_blocking_time** (p. 1298)
	<<***basic***>> *(p. 201)* <<***spec***>> *(p. 209)* The maximum time a DataWriter can block on a write() call.
DDS_PublishModeQosPolicy (p. 1279)	**kind** (p. 1281)
	<<***basic***>> *(p. 201)* <<***extension***>> *(p. 213)* Publishing mode

DDS_RtpsReliableWriterProtocol_t (p. 1320)	**min_send_window_size** (p. 1333)
	<<***basic***>> *(p. 201)* <<***extension***>> *(p. 213)* Minimum size of send window of unacknowledged samples
DDS_RtpsReliableWriterProtocol_t (p. 1320)	**max_send_window_size** (p. 1333)
	<<***basic***>> *(p. 201)* <<***extension***>> *(p. 213)* Maximum size of send window of unacknowledged samples

Advanced Settings

DDS_PublishModeQosPolicy (p. 1279)	**flow_controller_name** (p. 1281)
	<<***extension***>> *(p. 213)* Name of the associated flow controller
DDS_PublishModeQosPolicy (p. 1279)	**priority** (p. 1282)
	<<***extension***>> *(p. 213)* Publication priority
DDS_AsynchronousPublisherQosPolicy (p. 798)	**disable_asynchronous_write** (p. 800)
	[DEPRECATED] (p. **384**) <<***extension***>> *(p. 213)* Disable asynchronous publishing.
DDS_DomainParticipantResourceLimitsQosPolicy (p. 977)	**outstanding_asynchronous_sample_allocation** (p. 985)
	<<***extension***>> *(p. 213)* Allocation settings applied to the maximum number of samples (from all **DDSData↩ Writer** (p. 1589)) waiting to be asynchronously written.
DDS_DomainParticipantResourceLimitsQosPolicy (p. 977)	**flow_controller_allocation** (p. 986)
	<<***extension***>> *(p. 213)* Allocation settings applied to flow controllers.
DDS_DomainParticipantResourceLimitsQosPolicy (p. 977)	**flow_controller_hash_buckets** (p. 990)
	[DEPRECATED] (p. **384**) <<***extension***>> *(p. 213)* Number of hash buckets for flow controllers.
DDS_DiscoveryConfigQosPolicy (p. 934)	**publication_writer_publish_mode** (p. 948)
	<<***extension***>> *(p. 213)* Publish mode policy for the built-in publication writer.
DDS_DiscoveryConfigQosPolicy (p. 934)	**subscription_writer_publish_mode** (p. 949)
	<<***extension***>> *(p. 213)* Publish mode policy for the built-in subscription writer.
DDS_DiscoveryConfigQosPolicy (p. 934)	**asynchronous_publisher** (p. 949)

	<<***extension***>> *(p. 213)* Asynchronous publishing settings for the discovery **DDSPublisher** (p. 1818) and all entities that are created by it.
DDS_DiscoveryConfigQosPolicy (p. 934)	**service_request_writer_publish_mode** (p. 951)
	<<***extension***>> *(p. 213)* Publish mode policy for the built-in service request writer.
DDS_DiscoveryConfigQosPolicy (p. 934)	**secure_volatile_writer_publish_mode** (p. 955)
	<<***extension***>> *(p. 213)* Publish mode policy for the built-in secure volatile writer.
DDS_DiscoveryConfigQosPolicy (p. 934)	**participant_configuration_writer_publish_mode** (p. 957)
	<<***extension***>> *(p. 213)* Publish mode policy for the built-in participant configuration writer.
DDS_RtpsReliableWriterProtocol_t (p. 1320)	**max_bytes_per_nack_response** (p. 1329)
	<<***extension***>> *(p. 213)* The maximum total message size when resending rejected samples or data fragments.
DDS_RtpsReliableWriterProtocol_t (p. 1320)	**send_window_update_period** (p. 1334)
	<<***extension***>> *(p. 213)* Period in which send window may be dynamically changed
DDS_RtpsReliableWriterProtocol_t (p. 1320)	**send_window_increase_factor** (p. 1335)
	<<***extension***>> *(p. 213)* Increases send window size by this percentage when reacting dynamically to network conditions
DDS_RtpsReliableWriterProtocol_t (p. 1320)	**send_window_decrease_factor** (p. 1336)
	<<***extension***>> *(p. 213)* Decreases send window size by this percentage when reacting dynamically to network conditions
DDS_RtpsReliableWriterProtocol_t (p. 1320)	**disable_repair_piggyback_heartbeat** (p. 1338)
	<<***extension***>> *(p. 213)* Prevents piggyback heartbeats from being sent with repair samples

7.63 Fragmentation

QoS related to DDS-level data fragmentation.

QoS related to DDS-level data fragmentation.

See `Data Fragmentation`, in the *Core Libraries User's Manual*, for more information about DDS-level fragmentation.

Advanced Settings

DDS_DataReaderResourceLimitsQosPolicy (p. 867)	**disable_fragmentation_support** (p. 874)
	[DEPRECATED] (p. 384) <<***extension***>> *(p. 213)* Determines whether the **DDSDataReader** (p. 1556) can receive fragmented samples.
DDS_DataReaderResourceLimitsQosPolicy (p. 867)	**max_fragmented_samples** (p. 874)
	<<***extension***>> *(p. 213)* The maximum number of samples for which the **DDSDataReader** (p. 1556) may store fragments at a given point in time.
DDS_DataReaderResourceLimitsQosPolicy (p. 867)	**initial_fragmented_samples** (p. 875)
	<<***extension***>> *(p. 213)* The initial number of samples for which a **DDSDataReader** (p. 1556) may store fragments.
DDS_DataReaderResourceLimitsQosPolicy (p. 867)	**max_fragmented_samples_per_remote_writer** (p. 875)
	<<***extension***>> *(p. 213)* The maximum number of samples per remote writer for which a **DDSDataReader** (p. 1556) may store fragments.
DDS_DataReaderResourceLimitsQosPolicy (p. 867)	**max_fragments_per_sample** (p. 876)
	<<***extension***>> *(p. 213)* Maximum number of fragments for a single sample.
DDS_DataReaderResourceLimitsQosPolicy (p. 867)	**dynamically_allocate_fragmented_samples** (p. 876)
	<<***extension***>> *(p. 213)* Determines whether the **DDSDataReader** (p. 1556) pre-allocates storage for storing fragmented samples.
DDS_RtpsReliableWriterProtocol_t (p. 1320)	**min_nack_response_delay** (p. 1328)
	<<***extension***>> *(p. 213)* The minimum delay to respond to a NACK or NACK_FRAG.
DDS_RtpsReliableWriterProtocol_t (p. 1320)	**max_nack_response_delay** (p. 1328)
	<<***extension***>> *(p. 213)* The maximum delay to respond to a NACK or NACK_FRAG.
DDS_RtpsReliableWriterProtocol_t (p. 1320)	**nack_suppression_duration** (p. 1329)
	<<***extension***>> *(p. 213)* The duration for ignoring consecutive NACKs/NACK_FRAGs that may trigger redundant repairs.
DDS_RtpsReliableWriterProtocol_t (p. 1320)	**max_bytes_per_nack_response** (p. 1329)
	<<***extension***>> *(p. 213)* The maximum total message size when resending rejected samples or data fragments.
DDS_BuiltinTopicReaderResourceLimits_t (p. 815)	**disable_fragmentation_support** (p. 818)

	[DEPRECATED] (p. **384)** <<***extension***>> *(*p. *213)* Determines whether the built-in topic **DDSDataReader** (p. 1556) can receive fragmented samples.
DDS_BuiltinTopicReaderResourceLimits_t (p. 815)	**max_fragmented_samples** (p. 819)
	<<***extension***>> *(*p. *213)* The maximum number of samples for which the built-in topic **DDSDataReader** (p. 1556) may store fragments at a given point in time.
DDS_BuiltinTopicReaderResourceLimits_t (p. 815)	**initial_fragmented_samples** (p. 819)
	<<***extension***>> *(*p. *213)* The initial number of samples for which a built-in topic **DDSDataReader** (p. 1556) may store fragments.
DDS_BuiltinTopicReaderResourceLimits_t (p. 815)	**max_fragmented_samples_per_remote_writer** (p. 820)
	<<***extension***>> *(*p. *213)* The maximum number of samples per remote writer for which a built-in topic **DDSDataReader** (p. 1556) may store fragments.
DDS_BuiltinTopicReaderResourceLimits_t (p. 815)	**max_fragments_per_sample** (p. 820)
	<<***extension***>> *(*p. *213)* Maximum number of fragments for a single sample.
DDS_BuiltinTopicReaderResourceLimits_t (p. 815)	**dynamically_allocate_fragmented_samples** (p. 821)
	<<***extension***>> *(*p. *213)* Determines whether the built-in topic **DDSDataReader** (p. 1556) pre-allocates storage for storing fragmented samples.

7.64 Instance Lifecycle

QoS that control the lifecycle of instances.

QoS that control the lifecycle of instances.

Use these QoS fields to configure how RTI Connext manages instance lifecycles and resources in the DataReader and DataWriter queues.

Basic Settings

DDS_ResourceLimitsQosPolicy (p. 1308)	**max_instances** (p. 1311)
	<<***basic***>> *(*p. *201)* <<***spec***>> *(*p. *209)* Represents the maximum number of instances a **DDSDataWriter** (p. 1589) (or **DDSData↩ Reader** (p. 1556)) can manage.
DDS_ResourceLimitsQosPolicy (p. 1308)	**initial_instances** (p. 1312)

	<<***basic***>> *(p. 201)* <<***extension***>> *(p. 213)* Represents the initial number of instances a **DDSDataWriter** (p. 1589) (or **DDSData↩ Reader** (p. 1556)) will manage.

Advanced Settings

DDS_DurabilityServiceQosPolicy (p. 1009)	**service_cleanup_delay** (p. 1010)
	<<***spec***>> *(p. 209)* Controls when the service is able to remove all information regarding a data instances.
DDS_DurabilityServiceQosPolicy (p. 1009)	**max_instances** (p. 1011)
	<<***spec***>> *(p. 209)* Part of resource limits QoS policy to apply when feeding a late joiner.
DDS_DurabilityServiceQosPolicy (p. 1009)	**max_samples_per_instance** (p. 1012)
	<<***spec***>> *(p. 209)* Part of resource limits QoS policy to apply when feeding a late joiner.
DDS_ReliabilityQosPolicy (p. 1295)	**instance_state_consistency_kind** (p. 1299)
	<<***extension***>> *(p. 213)* Whether instance state consistency is enabled
DDS_ResourceLimitsQosPolicy (p. 1308)	**max_samples_per_instance** (p. 1311)
	<<***spec***>> *(p. 209)* Represents the maximum number of samples of any one instance a **DDSDataWriter** (p. 1589) (or **DDSDataReader** (p. 1556)) can manage.
DDS_ResourceLimitsQosPolicy (p. 1308)	**instance_hash_buckets** (p. 1313)
	[DEPRECATED] (p. **384)** <<***extension***>> *(p. 213)* Number of hash buckets for instances.
DDS_WriterDataLifecycleQosPolicy (p. 1524)	**autodispose_unregistered_instances** (p. 1525)
	<<***spec***>> *(p. 209)* Boolean flag that controls whether or not a writer will also dispose of an instance when the **DDSDataWriter** (p. 1589) unregisters from the instance by means of the unregister operations.
DDS_WriterDataLifecycleQosPolicy (p. 1524)	**autopurge_unregistered_instances_delay** (p. 1525)
	<<***extension***>> *(p. 213)* Maximum duration for which the **DDSDataWriter** (p. 1589) will maintain information regarding an instance once it has unregistered the instance.
DDS_WriterDataLifecycleQosPolicy (p. 1524)	**autopurge_disposed_instances_delay** (p. 1526)
	<<***extension***>> *(p. 213)* Maximum duration for which the **DDSDataWriter** (p. 1589) will maintain information regarding an instance once it has disposed the instance.

DDS_ReaderDataLifecycleQosPolicy (p. 1289)	**autopurge_nowriter_samples_delay** (p. 1291)
	<<***spec***>> *(p. 209)* Minimum duration for which the **DDSDataReader** (p. 1556) will maintain samples for an instance once its `instance_state` becomes **DDS_NOT_ALIVE_NO_WRITERS_INSTANCE_STATE** (p. 166).
DDS_ReaderDataLifecycleQosPolicy (p. 1289)	**autopurge_disposed_samples_delay** (p. 1291)
	<<***spec***>> *(p. 209)* Minimum duration for which the **DDSDataReader** (p. 1556) will maintain samples for an instance once its `instance_state` becomes **DDS_NOT_ALIVE_DISPOSED_INSTANCE_STATE** (p. 166).
DDS_ReaderDataLifecycleQosPolicy (p. 1289)	**autopurge_disposed_instances_delay** (p. 1291)
	<<***extension***>> *(p. 213)* Minimum duration for which the **DDSDataReader** (p. 1556) will maintain information about a received instance once its `instance_state` becomes **DDS_NOT_ALIVE_DISPOSED_INSTANCE_STATE** (p. 166) and there are no samples for the instance in the DataReader queue.
DDS_ReaderDataLifecycleQosPolicy (p. 1289)	**autopurge_nowriter_instances_delay** (p. 1292)
	<<***extension***>> *(p. 213)* Minimum duration for which the **DDSDataReader** (p. 1556) will maintain information about a received instance once its `instance_state` becomes **DDS_NOT_ALIVE_NO_WRITERS_INSTANCE_STATE** (p. 166) and there are no samples for the instance in the DataReader queue.
DDS_DataReaderResourceLimitsQosPolicy (p. 867)	**max_total_instances** (p. 877)
	<<***extension***>> *(p. 213)* Maximum number of instances for which a DataReader will keep state.
DDS_DataReaderResourceLimitsQosPolicy (p. 867)	**keep_minimum_state_for_instances** (p. 881)
	<<***extension***>> *(p. 213)* Whether or not keep a minimum instance state for up to **DDS_DataReaderResourceLimitsQosPolicy::max_total_instances** (p. 877).
DDS_DataReaderResourceLimitsQosPolicy (p. 867)	**instance_replacement** (p. 883)
	<<***extension***>> *(p. 213)* Sets the kind of instances allowed to be replaced for each instance state (**DDS_InstanceStateKind** (p. 165)) when a DataReader reaches **DDS_ResourceLimitsQosPolicy::max_instances** (p. 1311).
DDS_DataReaderResourceLimitsQosPolicy (p. 867)	**autopurge_remote_not_alive_writer_delay** (p. 884)

	<<***extension***>> *(p. 213)* Maximum duration for which the **DDSDataReader** (p. 1556) will maintain information regarding a **DDSDataWriter** (p. 1589) once the **DDSDataWriter** (p. 1589) has become not alive.
DDS_DataWriterResourceLimitsQosPolicy (p. 917)	**instance_replacement** (p. 921)
	<<***extension***>> *(p. 213)* Sets the kinds of instances allowed to be replaced when instance resource limits are reached.
DDS_DataWriterResourceLimitsQosPolicy (p. 917)	**replace_empty_instances** (p. 922)
	<<***extension***>> *(p. 213)* Whether or not to replace empty instances during instance replacement
DDS_DataWriterResourceLimitsQosPolicy (p. 917)	**autoregister_instances** (p. 922)
	[DEPRECATED] (p. 384) <<***extension***>> *(p. 213)* Whether or not to automatically register new instances
DDS_DataReaderProtocolQosPolicy (p. 843)	**propagate_dispose_of_unregistered_instances** (p. 845)
	<<***extension***>> *(p. 213)* Indicates whether or not an instance can move to the **DDS_NOT_ALIVE_↩ DISPOSED_INSTANCE_STATE** (p. 166) state without being in the **DDS_ALIVE_INSTANCE_STATE** (p. 166) state.
DDS_DataReaderProtocolQosPolicy (p. 843)	**propagate_unregister_of_disposed_instances** (p. 846)
	<<***extension***>> *(p. 213)* Indicates whether or not an instance can move to the **DDS_NOT_ALIVE_NO↩ _WRITERS_INSTANCE_STATE** (p. 166) state directly from the **DDS_NOT_ALIVE_DISPOSED_INSTANCE↩ _STATE** (p. 166).
DDS_DataWriterProtocolQosPolicy (p. 892)	**disable_inline_keyhash** (p. 895)
	<<***extension***>> *(p. 213)* Controls whether or not a keyhash is propagated on the wire with each sample.
DDS_DataWriterProtocolQosPolicy (p. 892)	**serialize_key_with_dispose** (p. 895)
	<<***extension***>> *(p. 213)* Controls whether or not the serialized key is propagated on the wire with dispose samples.
DDS_DiscoveryConfigQosPolicy (p. 934)	**publication_writer_data_lifecycle** (p. 944)
	<<***extension***>> *(p. 213)* Writer data lifecycle settings for a built-in publication writer.
DDS_DiscoveryConfigQosPolicy (p. 934)	**subscription_writer_data_lifecycle** (p. 945)
	<<***extension***>> *(p. 213)* Writer data lifecycle settings for a built-in subscription writer.

DDS_DiscoveryConfigQosPolicy (p. 934)	**service_request_writer_data_lifecycle** (p. 951)
	<<***extension***>> *(p. 213)* Writer data lifecycle settings for a built-in **DDS_ServiceRequest** (p. 1359) writer.
DDS_DiscoveryConfigQosPolicy (p. 934)	**participant_configuration_writer_data_lifecycle** (p. 958)
	<<***extension***>> *(p. 213)* Writer data lifecycle settings for a built-in participant configuration writer.

7.65 Sending Large Data

Use these QoS fields to optimize the process of sending large data.

Use these QoS fields to optimize the process of sending large data.

This includes QoS which help to reduce the number of copies needed and manage fragmentation.

See `Sending Large Data`, in the *Core Libraries User's Manual*, for more information.

Basic Settings

DDS_DataRepresentationQosPolicy (p. 886)	**compression_settings** (p. 888)
	<<***basic***>> *(p. 201)* <<***extension***>> *(p. 213)* Structure that contains the compression settings.

Advanced Settings

DDS_CompressionSettings_t (p. 826)	**compression_ids** (p. 826)
	<<***extension***>> *(p. 213)* Mask that represents the compression algorithms enabled.
DDS_CompressionSettings_t (p. 826)	**writer_compression_level** (p. 827)
	<<***extension***>> *(p. 213)* The level of compression to use when compressing data.
DDS_CompressionSettings_t (p. 826)	**writer_compression_threshold** (p. 827)
	<<***extension***>> *(p. 213)* The threshold, in bytes, above which a serialized sample will be eligible to be compressed.
DDS_DataReaderResourceLimitsQosPolicy (p. 867)	**disable_fragmentation_support** (p. 874)
	[DEPRECATED] (p. 384) <<***extension***>> *(p. 213)* Determines whether the **DDSDataReader** (p. 1556) can receive fragmented samples.

DDS_DataReaderResourceLimitsQosPolicy (p. 867)	**max_fragmented_samples** (p. 874)
	<<***extension***>> *(p. 213)* The maximum number of samples for which the **DDSDataReader** (p. 1556) may store fragments at a given point in time.
DDS_DataReaderResourceLimitsQosPolicy (p. 867)	**initial_fragmented_samples** (p. 875)
	<<***extension***>> *(p. 213)* The initial number of samples for which a **DDSDataReader** (p. 1556) may store fragments.
DDS_DataReaderResourceLimitsQosPolicy (p. 867)	**max_fragmented_samples_per_remote_writer** (p. 875)
	<<***extension***>> *(p. 213)* The maximum number of samples per remote writer for which a **DDSDataReader** (p. 1556) may store fragments.
DDS_DataReaderResourceLimitsQosPolicy (p. 867)	**max_fragments_per_sample** (p. 876)
	<<***extension***>> *(p. 213)* Maximum number of fragments for a single sample.
DDS_DataReaderResourceLimitsQosPolicy (p. 867)	**dynamically_allocate_fragmented_samples** (p. 876)
	<<***extension***>> *(p. 213)* Determines whether the **DDSDataReader** (p. 1556) pre-allocates storage for storing fragmented samples.
DDS_DataReaderResourceLimitsQosPolicy (p. 867)	**shmem_ref_transfer_mode_attached_segment_↩ allocation** (p. 883)
	<<***extension***>> *(p. 213)* Allocation resource for the shared memory segments attached by the **DDSData↩ Reader** (p. 1556).
DDS_DataWriterResourceLimitsQosPolicy (p. 917)	**writer_loaned_sample_allocation** (p. 925)
	<<***extension***>> *(p. 213)* Represents the allocation settings of loaned samples managed by a **DDSData↩ Writer** (p. 1589).
DDS_DataWriterResourceLimitsQosPolicy (p. 917)	**initialize_writer_loaned_sample** (p. 925)
	<<***extension***>> *(p. 213)* Whether or not to initialize loaned samples returned by a **DDSDataWriter** (p. 1589)
DDS_DataWriterTransferModeQosPolicy (p. 927)	**shmem_ref_settings** (p. 928)
	<<***extension***>> *(p. 213)* Settings related to transferring data using shared memory references.
DDS_DomainParticipantResourceLimitsQosPolicy (p. 977)	**shmem_ref_transfer_mode_max_segments** (p. 1004)
	<<***extension***>> *(p. 213)* Maximum number of segments created by all DataWriters belonging to a **DDSDomainParticipant** (p. 1619).

DDS_DiscoveryConfigQosPolicy (p. 934)	**endpoint_type_object_lb_serialization_threshold** (p. 956)
	<<***extension***>> *(p. 213)* Controls when/if the Type↩ Object v1 is compressed when propagated in Simple Endpoint Discovery.
DDS_BuiltinTopicReaderResourceLimits_t (p. 815)	**disable_fragmentation_support** (p. 818)
	[DEPRECATED] (p. 384) <<***extension***>> *(p. 213)* Determines whether the built-in topic **DDSDataReader** (p. 1556) can receive fragmented samples.
DDS_BuiltinTopicReaderResourceLimits_t (p. 815)	**max_fragmented_samples** (p. 819)
	<<***extension***>> *(p. 213)* The maximum number of samples for which the built-in topic **DDSDataReader** (p. 1556) may store fragments at a given point in time.
DDS_BuiltinTopicReaderResourceLimits_t (p. 815)	**initial_fragmented_samples** (p. 819)
	<<***extension***>> *(p. 213)* The initial number of samples for which a built-in topic **DDSDataReader** (p. 1556) may store fragments.
DDS_BuiltinTopicReaderResourceLimits_t (p. 815)	**max_fragmented_samples_per_remote_writer** (p. 820)
	<<***extension***>> *(p. 213)* The maximum number of samples per remote writer for which a built-in topic **DDSDataReader** (p. 1556) may store fragments.
DDS_BuiltinTopicReaderResourceLimits_t (p. 815)	**max_fragments_per_sample** (p. 820)
	<<***extension***>> *(p. 213)* Maximum number of fragments for a single sample.
DDS_BuiltinTopicReaderResourceLimits_t (p. 815)	**dynamically_allocate_fragmented_samples** (p. 821)
	<<***extension***>> *(p. 213)* Determines whether the built-in topic **DDSDataReader** (p. 1556) pre-allocates storage for storing fragmented samples.

7.66 Liveliness

Use these QoS fields to manage the liveliness of participants and endpoints, ensuring active communication and timely updates of the system state.

Use these QoS fields to manage the liveliness of participants and endpoints, ensuring active communication and timely updates of the system state.

Basic Settings

DDS_LivelinessQosPolicy (p. 1180)	**kind** (p. 1182)
	*<<**basic**>> (p. 201) <<**spec**>> (p. 209)* The kind of liveliness desired.
DDS_LivelinessQosPolicy (p. 1180)	**lease_duration** (p. 1182)
	*<<**basic**>> (p. 201) <<**spec**>> (p. 209)* The duration within which a **DDSDataWriter** (p. 1589) must be asserted, or else it is assumed to be not alive.
DDS_LivelinessQosPolicy (p. 1180)	**assertions_per_lease_duration** (p. 1183)
	*<<**basic**>> (p. 201) <<**extension**>> (p. 213)* The number of assertions a **DDSDataWriter** (p. 1589) will send during a its **DDS_↩ LivelinessQosPolicy::lease_duration** (p. 1182).
DDS_DiscoveryConfigQosPolicy (p. 934)	**participant_liveliness_lease_duration** (p. 937)
	*<<**basic**>> (p. 201) <<**extension**>> (p. 213)* The liveliness lease duration for the participant.
DDS_DiscoveryConfigQosPolicy (p. 934)	**participant_liveliness_assert_period** (p. 937)
	*<<**basic**>> (p. 201) <<**extension**>> (p. 213)* The period to assert liveliness for the participant.

Advanced Settings

DDS_DataReaderResourceLimitsQosPolicy (p. 867)	**autopurge_remote_not_alive_writer_delay** (p. 884)
	*<<**extension**>> (p. 213)* Maximum duration for which the **DDSDataReader** (p. 1556) will maintain information regarding a **DDSDataWriter** (p. 1589) once the **DDSDataWriter** (p. 1589) has become not alive.
DDS_DiscoveryConfigQosPolicy (p. 934)	**remote_participant_purge_kind** (p. 938)
	[DEPRECATED] (p. 384) *<<**extension**>> (p. 213)* The participant's behavior for maintaining knowledge of remote participants (and their contained entities) with which discovery communication has been lost.
DDS_DiscoveryConfigQosPolicy (p. 934)	**max_liveliness_loss_detection_period** (p. 938)
	[DEPRECATED] (p. 384) *<<**extension**>> (p. 213)* The maximum amount of time between when a remote entity stops maintaining its liveliness and when the matched local entity realizes that fact.
DDS_DiscoveryConfigQosPolicy (p. 934)	**participant_message_writer** (p. 947)

	<<***extension***>> *(*p. *213)* RTPS protocol-related configuration settings for the RTPS reliable writer associated with the built-in participant message writer. This parameter only has effect if the matching participant message reader is configured with **DDS_RELIABLE_RELIABILITY_QOS** (p. 633) **DDS_ReliabilityQosPolicyKind** (p. 632).
DDS_DiscoveryConfigQosPolicy (p. 934)	**participant_message_reader_reliability_kind** (p. 946)
	[DEPRECATED] (p. **384)** <<***extension***>> *(*p. *213)* Reliability policy for a built-in participant message reader.
DDS_DiscoveryConfigQosPolicy (p. 934)	**participant_message_reader** (p. 947)
	<<***extension***>> *(*p. *213)* RTPS reliable reader protocol-related configuration settings for a built-in participant message reader. This parameter only has effect if **DDS_DiscoveryConfigQosPolicy::participant_message_reader_reliability_kind** (p. 946) is set to **DDS_RELIABLE_RELIABILITY_QOS** (p. 633).
DDS_RtpsReliableWriterProtocol_t (p. 1320)	**max_heartbeat_retries** (p. 1326)
	<<***extension***>> *(*p. *213)* The maximum number of *periodic* heartbeat retries before marking a remote reader as inactive.
DDS_RtpsReliableWriterProtocol_t (p. 1320)	**inactivate_nonprogressing_readers** (p. 1326)
	<<***extension***>> *(*p. *213)* Whether to treat remote readers as inactive when their NACKs do not progress

7.67 Logging

Use these QoS fields to configure the RTI Connext logging infrastructure.

Use these QoS fields to configure the RTI Connext logging infrastructure.

This includes settings for configuring the verbosity and directing the output.

See `Logging`, in the *Core Libraries User's Manual*, for more information.

Basic Settings

DDS_LoggingQosPolicy (p. 1188)	**verbosity** (p. 1189)
	<<***basic***>> *(*p. *201)* <<***extension***>> *(*p. *213)* The verbosities at which RTI Connext diagnostic information is logged.
DDS_LoggingQosPolicy (p. 1188)	**category** (p. 1189)

	*<<**basic**>> (p. 201) <<**extension**>> (p. 213)* Categories of logged messages.
DDS_LoggingQosPolicy (p. 1188)	**output_file** (p. 1189)
	*<<**basic**>> (p. 201) <<**extension**>> (p. 213)* Specifies the file to which log messages will be redirected to.
DDS_LoggingQosPolicy (p. 1188)	**output_file_suffix** (p. 1190)
	*<<**basic**>> (p. 201) <<**extension**>> (p. 213)* Sets the file suffix when logging to a set of files
DDS_MonitoringLoggingDistributionSettings (p. 1200)	**max_historical_logs** (p. 1202)
	*<<**basic**>> (p. 201) <<**extension**>> (p. 213)* The number of log messages that RTI Monitoring Library 2.0 will keep as history.
DDS_MonitoringLoggingDistributionSettings (p. 1200)	**publication_period** (p. 1203)
	*<<**basic**>> (p. 201) <<**extension**>> (p. 213)* Period at which the logging thread publishes log messages.
DDS_MonitoringLoggingForwardingSettings (p. 1204)	**middleware_forwarding_level** (p. 1204)
	*<<**basic**>> (p. 201) <<**extension**>> (p. 213)* Log messages with **NDDS_CONFIG_LOG_FACILITY_↩ MIDDLEWARE** (p. 727) and a log level more verbose than this value will not be distributed by RTI Monitoring Library 2.0.
DDS_MonitoringLoggingForwardingSettings (p. 1204)	**security_event_forwarding_level** (p. 1204)
	*<<**basic**>> (p. 201) <<**extension**>> (p. 213)* Log messages with **NDDS_CONFIG_LOG_FACILITY_↩ SECURITY_EVENT** (p. 727) and a log level more verbose than this value will not be distributed by RTI Monitoring Library 2.0.
DDS_MonitoringLoggingForwardingSettings (p. 1204)	**service_forwarding_level** (p. 1205)
	*<<**basic**>> (p. 201) <<**extension**>> (p. 213)* Log messages with **NDDS_CONFIG_LOG_FACILITY_↩ SERVICE** (p. 727) and a log level more verbose than this value will not be distributed by RTI Monitoring Library 2.0.
DDS_MonitoringLoggingForwardingSettings (p. 1204)	**user_forwarding_level** (p. 1205)

	<<***basic***>> *(p. 201)* <<***extension***>> *(p. 213)* Log messages with **NDDS_CONFIG_LOG_FACILITY_USER** (p. 727) and a log level more verbose than this value will not be distributed by RTI Monitoring Library 2.0.
DDS_MonitoringTelemetryData (p. 1213)	**logs** (p. 1213)
	<<***basic***>> *(p. 201)* <<***extension***>> *(p. 213)* **DDS_MonitoringLoggingForwardingSettings** (p. 1204) containing the **NDDS_Config_SyslogVerbosity** (p. 727) levels that will be forwarded for the different **NDDS_Config_LogFacility** (p. 727).

Advanced Settings

DDS_LoggingQosPolicy (p. 1188)	**print_format** (p. 1189)
	<<***extension***>> *(p. 213)* The format used to output RTI Connext diagnostic information.
DDS_LoggingQosPolicy (p. 1188)	**max_bytes_per_file** (p. 1190)
	<<***extension***>> *(p. 213)* Specifies the maximum number of bytes a single file can contain
DDS_LoggingQosPolicy (p. 1188)	**max_files** (p. 1191)
	<<***extension***>> *(p. 213)* Specifies the maximum number of files to create before overwritting the previous ones.
DDS_MonitoringLoggingDistributionSettings (p. 1200)	**concurrency_level** (p. 1201)
	<<***extension***>> *(p. 213)* Defines how concurrent the push of log messages to RTI Monitoring Library 2.0 is.
DDS_MonitoringLoggingDistributionSettings (p. 1200)	**datawriter_qos_profile_name** (p. 1202)
	<<***extension***>> *(p. 213)* The fully qualified name of the profile used to configure the **DDSDataWriter** (p. 1589) that distributes log messages.
DDS_MonitoringLoggingDistributionSettings (p. 1200)	**thread** (p. 1203)
	<<***extension***>> *(p. 213)* The settings of the logging thread.
DDS_MonitoringDistributionSettings (p. 1195)	**logging_settings** (p. 1197)
	<<***extension***>> *(p. 213)* Configures the distribution of logging messages.

7.68 Middleware Memory Management

Use these QoS fields to configure the memory usage of the middleware, including various initial and max limits.

Use these QoS fields to configure the memory usage of the middleware, including various initial and max limits.

Basic Settings

DDS_ResourceLimitsQosPolicy (p. 1308)	**max_samples** (p. 1311)
	*<<**basic**>> (p. 201) <<**spec**>> (p. 209)* Represents the maximum samples the middleware can store for any one **DDSDataWriter** (p. 1589) (or **DDSDataReader** (p. 1556)).
DDS_ResourceLimitsQosPolicy (p. 1308)	**max_instances** (p. 1311)
	*<<**basic**>> (p. 201) <<**spec**>> (p. 209)* Represents the maximum number of instances a **DDSDataWriter** (p. 1589) (or **DDSDataReader** (p. 1556)) can manage.
DDS_ResourceLimitsQosPolicy (p. 1308)	**initial_samples** (p. 1312)
	*<<**basic**>> (p. 201) <<**extension**>> (p. 213)* Represents the initial samples the middleware will store for any one **DDSDataWriter** (p. 1589) (or **DDSDataReader** (p. 1556)).
DDS_ResourceLimitsQosPolicy (p. 1308)	**initial_instances** (p. 1312)
	*<<**basic**>> (p. 201) <<**extension**>> (p. 213)* Represents the initial number of instances a **DDSDataWriter** (p. 1589) (or **DDSDataReader** (p. 1556)) will manage.
DDS_MonitoringLoggingDistributionSettings (p. 1200)	**max_historical_logs** (p. 1202)
	*<<**basic**>> (p. 201) <<**extension**>> (p. 213)* The number of log messages that RTI Monitoring Library 2.0 will keep as history.

Advanced Settings

DDS_PersistentStorageSettings (p. 1242)	**writer_memory_state** (p. 1248)
	*<<**extension**>> (p. 213)* Determines how much state will be kept in memory by the durable writer history in order to avoid accessing the persistent storage in disk.
DDS_DurabilityServiceQosPolicy (p. 1009)	**max_samples** (p. 1011)
	*<<**spec**>> (p. 209)* Part of resource limits QoS policy to apply when feeding a late joiner.

DDS_DurabilityServiceQosPolicy (p. 1009)	**max_instances** (p. 1011)
	<<***spec***>> *(p. 209)* Part of resource limits QoS policy to apply when feeding a late joiner.
DDS_DurabilityServiceQosPolicy (p. 1009)	**max_samples_per_instance** (p. 1012)
	<<***spec***>> *(p. 209)* Part of resource limits QoS policy to apply when feeding a late joiner.
DDS_ResourceLimitsQosPolicy (p. 1308)	**max_samples_per_instance** (p. 1311)
	<<***spec***>> *(p. 209)* Represents the maximum number of samples of any one instance a **DDSDataWriter** (p. 1589) (or **DDSDataReader** (p. 1556)) can manage.
DDS_DataReaderResourceLimitsQosPolicy (p. 867)	**max_remote_writers** (p. 870)
	<<***extension***>> *(p. 213)* The maximum number of remote writers from which a **DDSDataReader** (p. 1556) may read, including all instances.
DDS_DataReaderResourceLimitsQosPolicy (p. 867)	**max_remote_writers_per_instance** (p. 870)
	<<***extension***>> *(p. 213)* The maximum number of remote writers from which a **DDSDataReader** (p. 1556) may read a single instance.
DDS_DataReaderResourceLimitsQosPolicy (p. 867)	**max_samples_per_remote_writer** (p. 870)
	<<***extension***>> *(p. 213)* The maximum number of out-of-order samples from a given remote **DDSData↩ Writer** (p. 1589) that a **DDSDataReader** (p. 1556) may store when maintaining a reliable connection to the **DDSDataWriter** (p. 1589).
DDS_DataReaderResourceLimitsQosPolicy (p. 867)	**max_infos** (p. 871)
	<<***extension***>> *(p. 213)* The maximum number of info units that a **DDSDataReader** (p. 1556) can use to store **DDS_SampleInfo** (p. 1344).
DDS_DataReaderResourceLimitsQosPolicy (p. 867)	**initial_remote_writers** (p. 871)
	<<***extension***>> *(p. 213)* The initial number of remote writers from which a **DDSDataReader** (p. 1556) may read, including all instances.
DDS_DataReaderResourceLimitsQosPolicy (p. 867)	**initial_remote_writers_per_instance** (p. 872)
	<<***extension***>> *(p. 213)* The initial number of remote writers from which a **DDSDataReader** (p. 1556) may read a single instance.
DDS_DataReaderResourceLimitsQosPolicy (p. 867)	**initial_infos** (p. 872)
	<<***extension***>> *(p. 213)* The initial number of info units that a **DDSDataReader** (p. 1556) can have, which are used to store **DDS_SampleInfo** (p. 1344).

DDS_DataReaderResourceLimitsQosPolicy (p. 867)	**initial_outstanding_reads** (p. 873)
	<<***extension***>> *(p. 213)* The initial number of outstanding calls to read/take (or one of their variants) on the same **DDSDataReader** (p. 1556) for which memory has not been returned by calling **FooDataReader::return_loan** (p. 1940).
DDS_DataReaderResourceLimitsQosPolicy (p. 867)	**max_outstanding_reads** (p. 873)
	<<***extension***>> *(p. 213)* The maximum number of outstanding read/take calls (or one of their variants) on the same **DDSDataReader** (p. 1556) for which memory has not been returned by calling **FooDataReader::return_loan** (p. 1940).
DDS_DataReaderResourceLimitsQosPolicy (p. 867)	**max_samples_per_read** (p. 873)
	<<***extension***>> *(p. 213)* The maximum number of data samples that the application can receive from the middleware in a single call to **FooDataReader::read** (p. 1919) or **FooDataReader::take** (p. 1920). If more data exists in the middleware, the application will need to issue multiple read/take calls.
DDS_DataReaderResourceLimitsQosPolicy (p. 867)	**max_fragmented_samples** (p. 874)
	<<***extension***>> *(p. 213)* The maximum number of samples for which the **DDSDataReader** (p. 1556) may store fragments at a given point in time.
DDS_DataReaderResourceLimitsQosPolicy (p. 867)	**initial_fragmented_samples** (p. 875)
	<<***extension***>> *(p. 213)* The initial number of samples for which a **DDSDataReader** (p. 1556) may store fragments.
DDS_DataReaderResourceLimitsQosPolicy (p. 867)	**max_fragmented_samples_per_remote_writer** (p. 875)
	<<***extension***>> *(p. 213)* The maximum number of samples per remote writer for which a **DDSDataReader** (p. 1556) may store fragments.
DDS_DataReaderResourceLimitsQosPolicy (p. 867)	**max_fragments_per_sample** (p. 876)
	<<***extension***>> *(p. 213)* Maximum number of fragments for a single sample.
DDS_DataReaderResourceLimitsQosPolicy (p. 867)	**dynamically_allocate_fragmented_samples** (p. 876)
	<<***extension***>> *(p. 213)* Determines whether the **DDSDataReader** (p. 1556) pre-allocates storage for storing fragmented samples.
DDS_DataReaderResourceLimitsQosPolicy (p. 867)	**max_total_instances** (p. 877)
	<<***extension***>> *(p. 213)* Maximum number of instances for which a DataReader will keep state.

DDS_DataReaderResourceLimitsQosPolicy (p. 867)	**max_remote_virtual_writers** (p. 878)
	<<***extension***>> *(p. 213)* The maximum number of remote virtual writers from which a **DDSDataReader** (p. 1556) may read, including all instances.
DDS_DataReaderResourceLimitsQosPolicy (p. 867)	**initial_remote_virtual_writers** (p. 878)
	<<***extension***>> *(p. 213)* The initial number of remote virtual writers from which a **DDSDataReader** (p. 1556) may read, including all instances.
DDS_DataReaderResourceLimitsQosPolicy (p. 867)	**max_remote_virtual_writers_per_instance** (p. 879)
	<<***extension***>> *(p. 213)* The maximum number of virtual remote writers that can be associated with an instance.
DDS_DataReaderResourceLimitsQosPolicy (p. 867)	**initial_remote_virtual_writers_per_instance** (p. 879)
	<<***extension***>> *(p. 213)* The initial number of virtual remote writers per instance.
DDS_DataReaderResourceLimitsQosPolicy (p. 867)	**max_remote_writers_per_sample** (p. 880)
	<<***extension***>> *(p. 213)* The maximum number of remote writers allowed to write the same sample.
DDS_DataReaderResourceLimitsQosPolicy (p. 867)	**max_query_condition_filters** (p. 880)
	<<***extension***>> *(p. 213)* The maximum number of query condition filters a reader is allowed.
DDS_DataReaderResourceLimitsQosPolicy (p. 867)	**max_app_ack_response_length** (p. 881)
	<<***extension***>> *(p. 213)* Maximum length of application-level acknowledgment response data
DDS_DataReaderResourceLimitsQosPolicy (p. 867)	**keep_minimum_state_for_instances** (p. 881)
	<<***extension***>> *(p. 213)* Whether or not keep a minimum instance state for up to **DDS_DataReader↩ ResourceLimitsQosPolicy::max_total_instances** (p. 877).
DDS_DataReaderResourceLimitsQosPolicy (p. 867)	**initial_topic_queries** (p. 882)
	<<***extension***>> *(p. 213)* The initial number of Topic↩ Queries allocated by a **DDSDataReader** (p. 1556).
DDS_DataReaderResourceLimitsQosPolicy (p. 867)	**max_topic_queries** (p. 882)
	<<***extension***>> *(p. 213)* The maximum number of active TopicQueries that a **DDSDataReader** (p. 1556) can create.
DDS_DataReaderResourceLimitsQosPolicy (p. 867)	**shmem_ref_transfer_mode_attached_segment_↩ allocation** (p. 883)

	<<***extension***>> *(p. 213)* Allocation resource for the shared memory segments attached by the **DDSData↩ Reader** (p. 1556).
DDS_DataReaderResourceLimitsQosPolicy (p. 867)	**instance_replacement** (p. 883)
	<<***extension***>> *(p. 213)* Sets the kind of instances allowed to be replaced for each instance state (**DDS↩ _InstanceStateKind** (p. 165)) when a DataReader reaches **DDS_ResourceLimitsQosPolicy::max_↩ instances** (p. 1311).
DDS_DataReaderResourceLimitsQosPolicy (p. 867)	**autopurge_remote_not_alive_writer_delay** (p. 884)
	<<***extension***>> *(p. 213)* Maximum duration for which the **DDSDataReader** (p. 1556) will maintain information regarding a **DDSDataWriter** (p. 1589) once the **DDSDataWriter** (p. 1589) has become not alive.
DDS_DataReaderResourceLimitsQosPolicy (p. 867)	**lazily_allocate_batch_resources** (p. 885)
	<<***extension***>> *(p. 213)* Determines whether the **DDSDataReader** (p. 1556) pre-allocates resources for receiving batches of samples or lazily allocates them when needed.
DDS_DataWriterResourceLimitsQosPolicy (p. 917)	**initial_concurrent_blocking_threads** (p. 919)
	[DEPRECATED] (p. **384**) <<***extension***>> *(p. 213)* The initial number of threads that are allowed to concurrently block on write call on the same **DDSDataWriter** (p. 1589).
DDS_DataWriterResourceLimitsQosPolicy (p. 917)	**max_concurrent_blocking_threads** (p. 919)
	[DEPRECATED] (p. **384**) <<***extension***>> *(p. 213)* The maximum number of threads that are allowed to concurrently block on write call on the same **DDSDataWriter** (p. 1589).
DDS_DataWriterResourceLimitsQosPolicy (p. 917)	**max_remote_reader_filters** (p. 920)
	<<***extension***>> *(p. 213)* The maximum number of remote DataReaders for which the **DDSDataWriter** (p. 1589) will perform content-based filtering.
DDS_DataWriterResourceLimitsQosPolicy (p. 917)	**initial_batches** (p. 920)
	<<***extension***>> *(p. 213)* Represents the initial number of batches a **DDSDataWriter** (p. 1589) will manage.
DDS_DataWriterResourceLimitsQosPolicy (p. 917)	**max_batches** (p. 921)
	<<***extension***>> *(p. 213)* Represents the maximum number of batches a **DDSDataWriter** (p. 1589) will manage.
DDS_DataWriterResourceLimitsQosPolicy (p. 917)	**instance_replacement** (p. 921)

	<<***extension***>> *(p. 213)* Sets the kinds of instances allowed to be replaced when instance resource limits are reached.
DDS_DataWriterResourceLimitsQosPolicy (p. 917)	**replace_empty_instances** (p. 922)
	<<***extension***>> *(p. 213)* Whether or not to replace empty instances during instance replacement
DDS_DataWriterResourceLimitsQosPolicy (p. 917)	**max_virtual_writers** (p. 923)
	<<***extension***>> *(p. 213)* The maximum number of virtual writers supported by a **DDSDataWriter** (p. 1589)
DDS_DataWriterResourceLimitsQosPolicy (p. 917)	**initial_virtual_writers** (p. 923)
	<<***extension***>> *(p. 213)* The initial number of virtual writers supported by a **DDSDataWriter** (p. 1589)
DDS_DataWriterResourceLimitsQosPolicy (p. 917)	**initial_remote_readers** (p. 926)
	<<***extension***>> *(p. 213)* Controls the initial memory allocation for the matched remote readers of a **DDSDataWriter** (p. 1589).
DDS_DataWriterResourceLimitsQosPolicy (p. 917)	**max_remote_readers** (p. 923)
	<<***extension***>> *(p. 213)* The maximum number of remote readers supported by a **DDSDataWriter** (p. 1589)
DDS_DataWriterResourceLimitsQosPolicy (p. 917)	**max_app_ack_remote_readers** (p. 924)
	<<***extension***>> *(p. 213)* The maximum number of application-level acknowledging remote readers supported by a **DDSDataWriter** (p. 1589)
DDS_DataWriterResourceLimitsQosPolicy (p. 917)	**initial_active_topic_queries** (p. 924)
	<<***extension***>> *(p. 213)* Represents the initial number of active topic queries a **DDSDataWriter** (p. 1589) will manage.
DDS_DataWriterResourceLimitsQosPolicy (p. 917)	**max_active_topic_queries** (p. 924)
	<<***extension***>> *(p. 213)* Represents the maximum number of active topic queries a **DDSDataWriter** (p. 1589) will manage.
DDS_DataWriterResourceLimitsQosPolicy (p. 917)	**writer_loaned_sample_allocation** (p. 925)
	<<***extension***>> *(p. 213)* Represents the allocation settings of loaned samples managed by a **DDSData↩ Writer** (p. 1589).
DDS_SystemResourceLimitsQosPolicy (p. 1382)	**max_objects_per_thread** (p. 1383)
	[DEPRECATED] (p. **384)** <<***extension***>> *(p. 213)* The maximum number of objects that can be stored per thread for a **DDSDomainParticipantFactory** (p. 1694).

DDS_SystemResourceLimitsQosPolicy (p. 1382)	**initial_objects_per_thread** (p. 1383)
	<<***extension***>> *(p. 213)* The number of objects per thread for a **DDSDomainParticipantFactory** (p. 1694) for which infrastructure will initially be allocated.
DDS_DomainParticipantResourceLimitsQosPolicy (p. 977)	**local_writer_allocation** (p. 981)
	<<***extension***>> *(p. 213)* Allocation settings applied to local DataWriters.
DDS_DomainParticipantResourceLimitsQosPolicy (p. 977)	**local_reader_allocation** (p. 981)
	<<***extension***>> *(p. 213)* Allocation settings applied to local DataReaders.
DDS_DomainParticipantResourceLimitsQosPolicy (p. 977)	**local_publisher_allocation** (p. 981)
	<<***extension***>> *(p. 213)* Allocation settings applied to local Publisher.
DDS_DomainParticipantResourceLimitsQosPolicy (p. 977)	**local_subscriber_allocation** (p. 981)
	<<***extension***>> *(p. 213)* Allocation settings applied to local Subscriber.
DDS_DomainParticipantResourceLimitsQosPolicy (p. 977)	**local_topic_allocation** (p. 982)
	<<***extension***>> *(p. 213)* Allocation settings applied to local Topic.
DDS_DomainParticipantResourceLimitsQosPolicy (p. 977)	**remote_writer_allocation** (p. 982)
	<<***extension***>> *(p. 213)* Allocation settings applied to remote DataWriters.
DDS_DomainParticipantResourceLimitsQosPolicy (p. 977)	**remote_reader_allocation** (p. 982)
	<<***extension***>> *(p. 213)* Allocation settings applied to remote DataReaders.
DDS_DomainParticipantResourceLimitsQosPolicy (p. 977)	**remote_participant_allocation** (p. 983)
	<<***extension***>> *(p. 213)* Allocation settings applied to remote DomainParticipants.
DDS_DomainParticipantResourceLimitsQosPolicy (p. 977)	**matching_writer_reader_pair_allocation** (p. 983)
	<<***extension***>> *(p. 213)* Allocation settings applied to matching local writer and remote/local reader pairs.

DDS_DomainParticipantResourceLimitsQosPolicy (p. 977)	**matching_reader_writer_pair_allocation** (p. 983)
	<<***extension***>> *(p. 213)* Allocation settings applied to matching local reader and remote/local writer pairs.
DDS_DomainParticipantResourceLimitsQosPolicy (p. 977)	**ignored_entity_allocation** (p. 984)
	<<***extension***>> *(p. 213)* Allocation settings applied to ignored entities.
DDS_DomainParticipantResourceLimitsQosPolicy (p. 977)	**content_filtered_topic_allocation** (p. 984)
	<<***extension***>> *(p. 213)* Allocation settings applied to content filtered topic.
DDS_DomainParticipantResourceLimitsQosPolicy (p. 977)	**content_filter_allocation** (p. 984)
	<<***extension***>> *(p. 213)* Allocation settings applied to content filter.
DDS_DomainParticipantResourceLimitsQosPolicy (p. 977)	**read_condition_allocation** (p. 985)
	<<***extension***>> *(p. 213)* Allocation settings applied to read condition pool.
DDS_DomainParticipantResourceLimitsQosPolicy (p. 977)	**query_condition_allocation** (p. 985)
	<<***extension***>> *(p. 213)* Allocation settings applied to query condition pool.
DDS_DomainParticipantResourceLimitsQosPolicy (p. 977)	**outstanding_asynchronous_sample_allocation** (p. 985)
	<<***extension***>> *(p. 213)* Allocation settings applied to the maximum number of samples (from all **DDSData↩ Writer** (p. 1589)) waiting to be asynchronously written.
DDS_DomainParticipantResourceLimitsQosPolicy (p. 977)	**flow_controller_allocation** (p. 986)
	<<***extension***>> *(p. 213)* Allocation settings applied to flow controllers.
DDS_DomainParticipantResourceLimitsQosPolicy (p. 977)	**local_writer_hash_buckets** (p. 986)
	[DEPRECATED] (p. 384) <<***extension***>> *(p. 213)* Hash_Buckets settings applied to local DataWriters.
DDS_DomainParticipantResourceLimitsQosPolicy (p. 977)	**local_reader_hash_buckets** (p. 986)
	[DEPRECATED] (p. 384) <<***extension***>> *(p. 213)* Number of hash buckets for local DataReaders.

DDS_DomainParticipantResourceLimitsQosPolicy (p. 977)	**local_publisher_hash_buckets** (p. 987)
	[DEPRECATED] (p. **384**) <<***extension***>> (p. *213*) Number of hash buckets for local Publisher.
DDS_DomainParticipantResourceLimitsQosPolicy (p. 977)	**local_subscriber_hash_buckets** (p. 987)
	[DEPRECATED] (p. **384**) <<***extension***>> (p. *213*) Number of hash buckets for local Subscriber.
DDS_DomainParticipantResourceLimitsQosPolicy (p. 977)	**local_topic_hash_buckets** (p. 987)
	[DEPRECATED] (p. **384**) <<***extension***>> (p. *213*) Number of hash buckets for local Topic.
DDS_DomainParticipantResourceLimitsQosPolicy (p. 977)	**remote_writer_hash_buckets** (p. 988)
	[DEPRECATED] (p. **384**) <<***extension***>> (p. *213*) Number of hash buckets for remote DataWriters.
DDS_DomainParticipantResourceLimitsQosPolicy (p. 977)	**remote_reader_hash_buckets** (p. 988)
	[DEPRECATED] (p. **384**) <<***extension***>> (p. *213*) Number of hash buckets for remote DataReaders.
DDS_DomainParticipantResourceLimitsQosPolicy (p. 977)	**remote_participant_hash_buckets** (p. 988)
	[DEPRECATED] (p. **384**) <<***extension***>> (p. *213*) Number of hash buckets for remote DomainParticipants.
DDS_DomainParticipantResourceLimitsQosPolicy (p. 977)	**matching_writer_reader_pair_hash_buckets** (p. 989)
	[DEPRECATED] (p. **384**) <<***extension***>> (p. *213*) Number of hash buckets for matching local writer and remote/local reader pairs.
DDS_DomainParticipantResourceLimitsQosPolicy (p. 977)	**matching_reader_writer_pair_hash_buckets** (p. 989)
	[DEPRECATED] (p. **384**) <<***extension***>> (p. *213*) Number of hash buckets for matching local reader and remote/local writer pairs.
DDS_DomainParticipantResourceLimitsQosPolicy (p. 977)	**ignored_entity_hash_buckets** (p. 989)
	[DEPRECATED] (p. **384**) <<***extension***>> (p. *213*) Number of hash buckets for ignored entities.
DDS_DomainParticipantResourceLimitsQosPolicy (p. 977)	**content_filtered_topic_hash_buckets** (p. 990)
	[DEPRECATED] (p. **384**) <<***extension***>> (p. *213*) Number of hash buckets for content filtered topics.

DDS_DomainParticipantResourceLimitsQosPolicy (p. 977)	**content_filter_hash_buckets** (p. 990)
	[DEPRECATED] (p. **384**) <<***extension***>> (p. *213*) Number of hash buckets for content filters.
DDS_DomainParticipantResourceLimitsQosPolicy (p. 977)	**flow_controller_hash_buckets** (p. 990)
	[DEPRECATED] (p. **384**) <<***extension***>> (p. *213*) Number of hash buckets for flow controllers.
DDS_DomainParticipantResourceLimitsQosPolicy (p. 977)	**max_gather_destinations** (p. 991)
	<<***extension***>> (p. *213*) Maximum number of destinations per RTI Connext send.
DDS_DomainParticipantResourceLimitsQosPolicy (p. 977)	**participant_user_data_max_length** (p. 991)
	<<***extension***>> (p. *213*) Maximum length of user data in **DDS_DomainParticipantQos** (p. 971) and **DDS_ParticipantBuiltinTopicData** (p. 1229)
DDS_DomainParticipantResourceLimitsQosPolicy (p. 977)	**topic_data_max_length** (p. 991)
	<<***extension***>> (p. *213*) Maximum length of topic data in **DDS_TopicQos** (p. 1397), **DDS_TopicBuiltinTopicData** (p. 1391), **DDS_PublicationBuiltinTopicData** (p. 1264) and **DDS_SubscriptionBuiltinTopicData** (p. 1370)
DDS_DomainParticipantResourceLimitsQosPolicy (p. 977)	**publisher_group_data_max_length** (p. 992)
	<<***extension***>> (p. *213*) Maximum length of group data in **DDS_PublisherQos** (p. 1276) and **DDS_PublicationBuiltinTopicData** (p. 1264)
DDS_DomainParticipantResourceLimitsQosPolicy (p. 977)	**subscriber_group_data_max_length** (p. 992)
	<<***extension***>> (p. *213*) Maximum length of group data in **DDS_SubscriberQos** (p. 1367) and **DDS_SubscriptionBuiltinTopicData** (p. 1370)
DDS_DomainParticipantResourceLimitsQosPolicy (p. 977)	**writer_user_data_max_length** (p. 992)
	<<***extension***>> (p. *213*) Maximum length of user data in **DDS_DataWriterQos** (p. 909) and **DDS_PublicationBuiltinTopicData** (p. 1264)
DDS_DomainParticipantResourceLimitsQosPolicy (p. 977)	**reader_user_data_max_length** (p. 993)

	<<***extension***>> *(p. 213)* Maximum length of user data in **DDS_DataReaderQos** (p. 858) and **DDS_SubscriptionBuiltinTopicData** (p. 1370)
DDS_DomainParticipantResourceLimitsQosPolicy (p. 977)	**max_partitions** (p. 993)
	<<***extension***>> *(p. 213)* Maximum number of partition name strings allowable in a **DDS_PartitionQosPolicy** (p. 1240).
DDS_DomainParticipantResourceLimitsQosPolicy (p. 977)	**max_partition_cumulative_characters** (p. 993)
	<<***extension***>> *(p. 213)* Maximum number of combined characters allowable in all partition names in a **DDS_PartitionQosPolicy** (p. 1240).
DDS_DomainParticipantResourceLimitsQosPolicy (p. 977)	**type_code_max_serialized_length** (p. 994)
	[DEPRECATED] (p. 384) <<***extension***>> *(p. 213)* Maximum size of serialized string for type code.
DDS_DomainParticipantResourceLimitsQosPolicy (p. 977)	**type_object_max_serialized_length** (p. 994)
	<<***extension***>> *(p. 213)* The maximum length, in bytes, that the buffer to serialize a TypeObject v1 can consume.
DDS_DomainParticipantResourceLimitsQosPolicy (p. 977)	**type_object_max_deserialized_length** (p. 996)
	<<***extension***>> *(p. 213)* The maximum number of bytes that a deserialized TypeObject v1 can consume.
DDS_DomainParticipantResourceLimitsQosPolicy (p. 977)	**deserialized_type_object_dynamic_allocation_threshold** (p. 996)
	<<***extension***>> *(p. 213)* A threshold, in bytes, for dynamic memory allocation for the deserialized TypeObject v1.
DDS_DomainParticipantResourceLimitsQosPolicy (p. 977)	**serialized_type_object_dynamic_allocation_threshold** (p. 995)
	<<***extension***>> *(p. 213)* A threshold, in bytes, for dynamic memory allocation for the serialized TypeObject v1.
DDS_DomainParticipantResourceLimitsQosPolicy (p. 977)	**contentfilter_property_max_length** (p. 997)
	<<***extension***>> *(p. 213)* This field is the maximum length of all data related to a Content-filtered topic.
DDS_DomainParticipantResourceLimitsQosPolicy (p. 977)	**channel_seq_max_length** (p. 997)

	<<***extension***>> *(p. 213)* Maximum number of channels that can be specified in **DDS_MultiChannelQos↩ Policy** (p. 1214) for MultiChannel DataWriters.
DDS_DomainParticipantResourceLimitsQosPolicy (p. 977)	**channel_filter_expression_max_length** (p. 998)
	<<***extension***>> *(p. 213)* Maximum length of a channel **DDS_ChannelSettings_t::filter_expression** (p. 822) in a MultiChannel DataWriter.
DDS_DomainParticipantResourceLimitsQosPolicy (p. 977)	**participant_property_list_max_length** (p. 998)
	<<***extension***>> *(p. 213)* Maximum number of properties associated with the **DDSDomainParticipant** (p. 1619).
DDS_DomainParticipantResourceLimitsQosPolicy (p. 977)	**participant_property_string_max_length** (p. 998)
	<<***extension***>> *(p. 213)* Maximum string length of the properties associated with the **DDSDomainParticipant** (p. 1619).
DDS_DomainParticipantResourceLimitsQosPolicy (p. 977)	**writer_property_list_max_length** (p. 999)
	<<***extension***>> *(p. 213)* Maximum number of properties associated with a **DDSDataWriter** (p. 1589).
DDS_DomainParticipantResourceLimitsQosPolicy (p. 977)	**writer_property_string_max_length** (p. 999)
	<<***extension***>> *(p. 213)* Maximum string length of the properties associated with a **DDSDataWriter** (p. 1589).
DDS_DomainParticipantResourceLimitsQosPolicy (p. 977)	**reader_property_list_max_length** (p. 999)
	<<***extension***>> *(p. 213)* Maximum number of properties associated with a **DDSDataReader** (p. 1556).
DDS_DomainParticipantResourceLimitsQosPolicy (p. 977)	**reader_property_string_max_length** (p. 1000)
	<<***extension***>> *(p. 213)* Maximum string length of the properties associated with a **DDSDataReader** (p. 1556).
DDS_DomainParticipantResourceLimitsQosPolicy (p. 977)	**max_endpoint_groups** (p. 1000)
	<<***extension***>> *(p. 213)* Maximum number of **DDS↩ _EndpointGroup_t** (p. 1142) allowable in a **DDS_↩ AvailabilityQosPolicy** (p. 805).
DDS_DomainParticipantResourceLimitsQosPolicy (p. 977)	**max_endpoint_group_cumulative_characters** (p. 1000)

	<<***extension***>> *(p. 213)* Maximum number of combined role_name characters allowed in all **DDS_EndpointGroup_t** (p. 1142) in a **DDS_AvailabilityQosPolicy** (p. 805).
DDS_DomainParticipantResourceLimitsQosPolicy (p. 977)	**transport_info_list_max_length** (p. 1001)
	<<***extension***>> *(p. 213)* Maximum number of installed transports to send and receive information about in **DDS_ParticipantBuiltinTopicData::transport_info** (p. 1232).
DDS_DomainParticipantResourceLimitsQosPolicy (p. 977)	**ignored_entity_replacement_kind** (p. 1001)
	[DEPRECATED] (p. **384**) <<***extension***>> *(p. 213)* Replacement policy for the ignored entities. It sets what entity can be replaced when resource limits set in **DDS_DomainParticipantResourceLimitsQosPolicy::ignored_entity_allocation** (p. 984) are reached.
DDS_DomainParticipantResourceLimitsQosPolicy (p. 977)	**remote_topic_query_allocation** (p. 1001)
	<<***extension***>> *(p. 213)* Allocation settings applied to remote TopicQueries.
DDS_DomainParticipantResourceLimitsQosPolicy (p. 977)	**remote_topic_query_hash_buckets** (p. 1002)
	[DEPRECATED] (p. **384**) <<***extension***>> *(p. 213)* Number of hash buckets for remote TopicQueries.
DDS_DomainParticipantResourceLimitsQosPolicy (p. 977)	**writer_data_tag_list_max_length** (p. 1002)
	<<***extension***>> *(p. 213)* Maximum number of data tags associated with a **DDSDataWriter** (p. 1589).
DDS_DomainParticipantResourceLimitsQosPolicy (p. 977)	**writer_data_tag_string_max_length** (p. 1003)
	<<***extension***>> *(p. 213)* Maximum string length of the data tags associated with a **DDSDataWriter** (p. 1589).
DDS_DomainParticipantResourceLimitsQosPolicy (p. 977)	**reader_data_tag_list_max_length** (p. 1003)
	<<***extension***>> *(p. 213)* Maximum number of data tags associated with a **DDSDataReader** (p. 1556).
DDS_DomainParticipantResourceLimitsQosPolicy (p. 977)	**reader_data_tag_string_max_length** (p. 1003)
	<<***extension***>> *(p. 213)* Maximum string length of the data tags associated with a **DDSDataReader** (p. 1556).

DDS_DomainParticipantResourceLimitsQosPolicy (p. 977)	**shmem_ref_transfer_mode_max_segments** (p. 1004)
	<<***extension***>> *(p. 213)* Maximum number of segments created by all DataWriters belonging to a **DDSDomainParticipant** (p. 1619).
DDS_EventQosPolicy (p. 1150)	**initial_count** (p. 1151)
	<<***extension***>> *(p. 213)* The initial number of events.
DDS_EventQosPolicy (p. 1150)	**max_count** (p. 1152)
	<<***extension***>> *(p. 213)* The maximum number of events.
DDS_DatabaseQosPolicy (p. 831)	**initial_records** (p. 833)
	<<***extension***>> *(p. 213)* The initial number of total records.
DDS_DatabaseQosPolicy (p. 831)	**max_skiplist_level** (p. 834)
	[DEPRECATED] (p. 384) <<***extension***>> *(p. 213)* The maximum level of the skiplist.
DDS_DatabaseQosPolicy (p. 831)	**initial_weak_references** (p. 835)
	<<***extension***>> *(p. 213)* The initial number of weak references.
DDS_DatabaseQosPolicy (p. 831)	**max_weak_references** (p. 834)
	<<***extension***>> *(p. 213)* The maximum number of weak references.
DDS_ReceiverPoolQosPolicy (p. 1293)	**buffer_size** (p. 1294)
	<<***extension***>> *(p. 213)* The receive buffer size in bytes.
DDS_ReceiverPoolQosPolicy (p. 1293)	**buffer_alignment** (p. 1295)
	[DEPRECATED] (p. 384) <<***extension***>> *(p. 213)* The receive buffer alignment.
DDS_DiscoveryConfigQosPolicy (p. 934)	**participant_reader_resource_limits** (p. 941)
	<<***extension***>> *(p. 213)* Resource limits.
DDS_DiscoveryConfigQosPolicy (p. 934)	**publication_reader_resource_limits** (p. 942)
	<<***extension***>> *(p. 213)* Resource limits.
DDS_DiscoveryConfigQosPolicy (p. 934)	**subscription_reader_resource_limits** (p. 943)
	<<***extension***>> *(p. 213)* Resource limits.
DDS_DiscoveryConfigQosPolicy (p. 934)	**participant_configuration_reader_resource_limits** (p. 959)
	<<***extension***>> *(p. 213)* Resource limits for the built-in topic participant configuration reader.

DDS_BuiltinTopicReaderResourceLimits_t (p. 815)	**initial_samples** (p. 816)
	<<***extension***>> *(p. 213)* Initial number of samples.
DDS_BuiltinTopicReaderResourceLimits_t (p. 815)	**max_samples** (p. 816)
	<<***extension***>> *(p. 213)* Maximum number of samples.
DDS_BuiltinTopicReaderResourceLimits_t (p. 815)	**initial_infos** (p. 817)
	<<***extension***>> *(p. 213)* Initial number of sample infos.
DDS_BuiltinTopicReaderResourceLimits_t (p. 815)	**max_infos** (p. 817)
	<<***extension***>> *(p. 213)* Maximum number of sample infos.
DDS_BuiltinTopicReaderResourceLimits_t (p. 815)	**initial_outstanding_reads** (p. 817)
	<<***extension***>> *(p. 213)* The initial number of outstanding reads that have not called finish yet on the same built-in topic **DDSDataReader** (p. 1556).
DDS_BuiltinTopicReaderResourceLimits_t (p. 815)	**max_outstanding_reads** (p. 818)
	<<***extension***>> *(p. 213)* The maximum number of outstanding reads that have not called finish yet on the same built-in topic **DDSDataReader** (p. 1556).
DDS_BuiltinTopicReaderResourceLimits_t (p. 815)	**max_samples_per_read** (p. 818)
	<<***extension***>> *(p. 213)* Maximum number of samples that can be read/taken on a same built-in topic **DDSDataReader** (p. 1556).
DDS_BuiltinTopicReaderResourceLimits_t (p. 815)	**disable_fragmentation_support** (p. 818)
	[DEPRECATED] (p. 384) <<***extension***>> *(p. 213)* Determines whether the built-in topic **DDSDataReader** (p. 1556) can receive fragmented samples.
DDS_BuiltinTopicReaderResourceLimits_t (p. 815)	**max_fragmented_samples** (p. 819)
	<<***extension***>> *(p. 213)* The maximum number of samples for which the built-in topic **DDSDataReader** (p. 1556) may store fragments at a given point in time.
DDS_BuiltinTopicReaderResourceLimits_t (p. 815)	**initial_fragmented_samples** (p. 819)
	<<***extension***>> *(p. 213)* The initial number of samples for which a built-in topic **DDSDataReader** (p. 1556) may store fragments.
DDS_BuiltinTopicReaderResourceLimits_t (p. 815)	**max_fragmented_samples_per_remote_writer** (p. 820)

	<<***extension***>> *(p. 213)* The maximum number of samples per remote writer for which a built-in topic **DDSDataReader** (p. 1556) may store fragments.
DDS_BuiltinTopicReaderResourceLimits_t (p. 815)	**max_fragments_per_sample** (p. 820)
	<<***extension***>> *(p. 213)* Maximum number of fragments for a single sample.
DDS_BuiltinTopicReaderResourceLimits_t (p. 815)	**dynamically_allocate_fragmented_samples** (p. 821)
	<<***extension***>> *(p. 213)* Determines whether the built-in topic **DDSDataReader** (p. 1556) pre-allocates storage for storing fragmented samples.
DDS_ThreadSettings_t (p. 1385)	**stack_size** (p. 1386)
	The thread stack-size.
DDS_LoggingQosPolicy (p. 1188)	**max_bytes_per_file** (p. 1190)
	<<***extension***>> *(p. 213)* Specifies the maximum number of bytes a single file can contain
DDS_LoggingQosPolicy (p. 1188)	**max_files** (p. 1191)
	<<***extension***>> *(p. 213)* Specifies the maximum number of files to create before overwritting the previous ones.

7.69 QoS Profile Management

Use these QoS fields to configure how QoS profiles are loaded and which ones should be used when entities are created.

Use these QoS fields to configure how QoS profiles are loaded and which ones should be used when entities are created.

Basic Settings

DDS_ProfileQosPolicy (p. 1257)	**string_profile** (p. 1258)
	<<***basic***>> *(p. 201)* <<***extension***>> *(p. 213)* Sequence of strings containing a XML document to load.
DDS_ProfileQosPolicy (p. 1257)	**url_profile** (p. 1258)
	<<***basic***>> *(p. 201)* <<***extension***>> *(p. 213)* Sequence of **URL groups** (p. 392) containing a set of XML documents to load.
DDS_ProfileQosPolicy (p. 1257)	**ignore_user_profile** (p. 1258)
	<<***basic***>> *(p. 201)* <<***extension***>> *(p. 213)* Ignores the file USER_QOS_PROFILES.xml in the current working directory.

DDS_ProfileQosPolicy (p. 1257)	**ignore_environment_profile** (p. 1258)
	*<<**basic**>> (p. 201) <<**extension**>> (p. 213)* Ignores the value of the **NDDS_QOS_PROFILES** (p. 392).
DDS_ProfileQosPolicy (p. 1257)	**ignore_resource_profile** (p. 1259)
	*<<**basic**>> (p. 201) <<**extension**>> (p. 213)* Ignores the file NDDS_QOS_PROFILES.xml.
DDS_MonitoringDedicatedParticipantSettings (p. 1193)	**enable** (p. 1193)
	*<<**basic**>> (p. 201) <<**extension**>> (p. 213)* Enables the use of a dedicated **DDSDomainParticipant** (p. 1619) to distribute the RTI Connext application telemetry data.

Advanced Settings

DDS_MonitoringEventDistributionSettings (p. 1197)	**datawriter_qos_profile_name** (p. 1199)
	*<<**extension**>> (p. 213)* The fully qualified name of the profile used to configure the **DDSDataWriter** (p. 1589) that distributes event metrics.
DDS_MonitoringPeriodicDistributionSettings (p. 1209)	**datawriter_qos_profile_name** (p. 1209)
	*<<**extension**>> (p. 213)* The fully qualified name of the profile used to configure the **DDSDataWriter** (p. 1589) that distributes periodic metrics.
DDS_MonitoringLoggingDistributionSettings (p. 1200)	**datawriter_qos_profile_name** (p. 1202)
	*<<**extension**>> (p. 213)* The fully qualified name of the profile used to configure the **DDSDataWriter** (p. 1589) that distributes log messages.
DDS_MonitoringDedicatedParticipantSettings (p. 1193)	**participant_qos_profile_name** (p. 1194)
	*<<**extension**>> (p. 213)* The fully qualified name of the profile used to configure the **DDSDomain↩ Participant** (p. 1619) that will be used to distribute telemetry data.
DDS_MonitoringDistributionSettings (p. 1195)	**dedicated_participant** (p. 1196)
	*<<**extension**>> (p. 213)* Configures the use of a dedicated **DDSDomainParticipant** (p. 1619) to distribute the RTI Connext application telemetry data.
DDS_MonitoringDistributionSettings (p. 1195)	**publisher_qos_profile_name** (p. 1196)

	<<***extension***>> *(p. 213)* The fully qualified name of the profile used to configure the Publishers that distribute telemetry data.

7.70 Reliability QoS

Reliability related QoS.

Reliability related QoS.

Use these QoS fields to configure the reliable delivery of your application data – this includes those settings that configure the reliability protocol.

See `Reliable Delivery Model`, in the *Core Libraries User's Manual*, for more information about the DDS reliability protocol.

Basic Settings

DDS_ReliabilityQosPolicy (p. 1295)	**kind** (p. 1298)
	<<***basic***>> *(p. 201)* <<***spec***>> *(p. 209)* Kind of reliability.
DDS_ReliabilityQosPolicy (p. 1295)	**max_blocking_time** (p. 1298)
	<<***basic***>> *(p. 201)* <<***spec***>> *(p. 209)* The maximum time a DataWriter can block on a write() call.
DDS_HistoryQosPolicy (p. 1162)	**kind** (p. 1164)
	<<***basic***>> *(p. 201)* <<***spec***>> *(p. 209)* Specifies the kind of history to be kept.
DDS_HistoryQosPolicy (p. 1162)	**depth** (p. 1164)
	<<***basic***>> *(p. 201)* <<***spec***>> *(p. 209)* Specifies the number of samples per instance to be kept, when the `kind` is **DDS_↩ KEEP_LAST_HISTORY_QOS** (p. 604)
DDS_RtpsReliableWriterProtocol_t (p. 1320)	**heartbeat_period** (p. 1323)
	<<***basic***>> *(p. 201)* <<***extension***>> *(p. 213)* The period at which to send heartbeats.
DDS_RtpsReliableWriterProtocol_t (p. 1320)	**fast_heartbeat_period** (p. 1323)
	<<***basic***>> *(p. 201)* <<***extension***>> *(p. 213)* An alternative heartbeat period used when a reliable writer needs to flush its unacknowledged samples more quickly.
DDS_RtpsReliableWriterProtocol_t (p. 1320)	**late_joiner_heartbeat_period** (p. 1324)

	<<***basic***>> *(*p. *201)* <<***extension***>> *(*p. *213)* An alternative heartbeat period used when a reliable reader joins late and needs to be caught up on cached samples of a reliable writer more quickly than the normal heartbeat rate.
DDS_RtpsReliableWriterProtocol_t (p. 1320)	**min_send_window_size** (p. 1333)
	<<***basic***>> *(*p. *201)* <<***extension***>> *(*p. *213)* Minimum size of send window of unacknowledged samples
DDS_RtpsReliableWriterProtocol_t (p. 1320)	**max_send_window_size** (p. 1333)
	<<***basic***>> *(*p. *201)* <<***extension***>> *(*p. *213)* Maximum size of send window of unacknowledged samples

Advanced Settings

DDS_AvailabilityQosPolicy (p. 805)	**enable_required_subscriptions** (p. 808)
	<<***extension***>> *(*p. *213)* Enables support for required subscriptions in a **DDSDataWriter** (p. 1589)
DDS_AvailabilityQosPolicy (p. 805)	**max_data_availability_waiting_time** (p. 808)
	<<***extension***>> *(*p. *213)* Defines how much time to wait before delivering a sample to the application without having received some of the previous samples.
DDS_AvailabilityQosPolicy (p. 805)	**max_endpoint_availability_waiting_time** (p. 808)
	<<***extension***>> *(*p. *213)* Defines how much time to wait to discover DataWriters providing samples for the same data source (virtual GUID).
DDS_AvailabilityQosPolicy (p. 805)	**required_matched_endpoint_groups** (p. 809)
	<<***extension***>> *(*p. *213)* A sequence of endpoint groups.
DDS_ReliabilityQosPolicy (p. 1295)	**acknowledgment_kind** (p. 1298)
	<<***extension***>> *(*p. *213)* Kind of reliable acknowledgment
DDS_ReliabilityQosPolicy (p. 1295)	**instance_state_consistency_kind** (p. 1299)
	<<***extension***>> *(*p. *213)* Whether instance state consistency is enabled
DDS_ReliabilityQosPolicy (p. 1295)	**exact_reliability_kind_match** (p. 1299)
	<<***extension***>> *(*p. *213)* Whether an exact reliability kind match is required for endpoint matching.
DDS_DataReaderResourceLimitsQosPolicy (p. 867)	**max_app_ack_response_length** (p. 881)
	<<***extension***>> *(*p. *213)* Maximum length of application-level acknowledgment response data

DDS_DataWriterResourceLimitsQosPolicy (p. 917)	**max_app_ack_remote_readers** (p. 924)
	<<***extension***>> *(p. 213)* The maximum number of application-level acknowledging remote readers supported by a **DDSDataWriter** (p. 1589)
DDS_DataReaderProtocolQosPolicy (p. 843)	**rtps_reliable_reader** (p. 846)
	<<***extension***>> *(p. 213)* RTPS protocol-related configuration settings for the RTPS reliable reader associated with a **DDSDataReader** (p. 1556). This parameter only has effect if the reader is configured with **DDS_RELIABLE_RELIABILITY_QOS** (p. 633) **DDS_↩ ReliabilityQosPolicyKind** (p. 632).
DDS_DataReaderProtocolQosPolicy (p. 843)	**disable_positive_acks** (p. 845)
	<<***extension***>> *(p. 213)* Whether the reader sends positive acknowledgements to writers.
DDS_DataWriterProtocolQosPolicy (p. 892)	**push_on_write** (p. 894)
	<<***extension***>> *(p. 213)* Whether to push sample out when write is called.
DDS_DataWriterProtocolQosPolicy (p. 892)	**disable_positive_acks** (p. 894)
	<<***extension***>> *(p. 213)* Controls whether or not the writer expects positive acknowledgements from matching readers.
DDS_DataWriterProtocolQosPolicy (p. 892)	**propagate_app_ack_with_no_response** (p. 896)
	<<***extension***>> *(p. 213)* Controls whether or not a **DDSDataWriter** (p. 1589) receives **DDSDataWriter↩ Listener::on_application_acknowledgment** (p. 1617) notifications with an empty or invalid response.
DDS_DataWriterProtocolQosPolicy (p. 892)	**rtps_reliable_writer** (p. 896)
	<<***extension***>> *(p. 213)* RTPS protocol-related configuration settings for the RTPS reliable writer associated with a **DDSDataWriter** (p. 1589). This parameter only has effect if both the writer and the matching reader are configured with **DDS_RELIABLE_RELIABILITY_↩ QOS** (p. 633) **DDS_ReliabilityQosPolicyKind** (p. 632).
DDS_DiscoveryConfigQosPolicy (p. 934)	**publication_reader** (p. 941)
	<<***extension***>> *(p. 213)* RTPS protocol-related configuration settings for the RTPS reliable reader associated with a built-in publication reader.
DDS_DiscoveryConfigQosPolicy (p. 934)	**subscription_reader** (p. 942)
	<<***extension***>> *(p. 213)* RTPS protocol-related configuration settings for the RTPS reliable reader associated with a built-in subscription reader.

DDS_DiscoveryConfigQosPolicy (p. 934)	**participant_message_writer** (p. 947)
	<<***extension***>> *(p. 213)* RTPS protocol-related configuration settings for the RTPS reliable writer associated with the built-in participant message writer. This parameter only has effect if the matching participant message reader is configured with **DDS_RELIABLE↩ _RELIABILITY_QOS** (p. 633) **DDS_ReliabilityQos↩ PolicyKind** (p. 632).
DDS_DiscoveryConfigQosPolicy (p. 934)	**participant_message_reader_reliability_kind** (p. 946)
	[DEPRECATED] (p. 384) <<***extension***>> *(p. 213)* Reliability policy for a built-in participant message reader.
DDS_DiscoveryConfigQosPolicy (p. 934)	**participant_message_reader** (p. 947)
	<<***extension***>> *(p. 213)* RTPS reliable reader protocol-related configuration settings for a built-in participant message reader. This parameter only has effect if **DDS_DiscoveryConfigQosPolicy::participant_↩ message_reader_reliability_kind** (p. 946) is set to **DDS_RELIABLE_RELIABILITY_QOS** (p. 633).
DDS_DiscoveryConfigQosPolicy (p. 934)	**publication_writer** (p. 943)
	<<***extension***>> *(p. 213)* RTPS protocol-related configuration settings for the RTPS reliable writer associated with a built-in publication writer.
DDS_DiscoveryConfigQosPolicy (p. 934)	**subscription_writer** (p. 945)
	<<***extension***>> *(p. 213)* RTPS protocol-related configuration settings for the RTPS reliable writer associated with a built-in subscription writer.
DDS_DiscoveryConfigQosPolicy (p. 934)	**service_request_writer** (p. 950)
	<<***extension***>> *(p. 213)* RTPS protocol-related configuration settings for the RTPS reliable writer associated with the built-in **DDS_ServiceRequest** (p. 1359) writer.
DDS_DiscoveryConfigQosPolicy (p. 934)	**service_request_reader** (p. 952)
	<<***extension***>> *(p. 213)* RTPS reliable reader protocol-related configuration settings for a built-in **DDS_ServiceRequest** (p. 1359) reader.
DDS_DiscoveryConfigQosPolicy (p. 934)	**secure_volatile_writer** (p. 954)
	<<***extension***>> *(p. 213)* RTPS protocol-related configuration settings for the RTPS reliable writer associated with the built-in secure volatile writer.
DDS_DiscoveryConfigQosPolicy (p. 934)	**secure_volatile_reader** (p. 955)
	<<***extension***>> *(p. 213)* RTPS reliable reader protocol-related configuration settings for the built-in secure volatile reader.

DDS_DiscoveryConfigQosPolicy (p. 934)	**participant_configuration_reader** (p. 959)
	<<***extension***>> *(p. 213)* RTPS protocol-related configuration settings for the RTPS reliable reader associated with a built-in participant configuration reader.
DDS_DiscoveryConfigQosPolicy (p. 934)	**participant_configuration_writer** (p. 957)
	<<***extension***>> *(p. 213)* RTPS protocol-related configuration settings for the RTPS reliable writer associated with a built-in participant configuration writer.
DDS_RtpsReliableReaderProtocol_t (p. 1314)	**min_heartbeat_response_delay** (p. 1315)
	<<***extension***>> *(p. 213)* The minimum delay to respond to a heartbeat.
DDS_RtpsReliableReaderProtocol_t (p. 1314)	**max_heartbeat_response_delay** (p. 1315)
	<<***extension***>> *(p. 213)* The maximum delay to respond to a heartbeat.
DDS_RtpsReliableReaderProtocol_t (p. 1314)	**heartbeat_suppression_duration** (p. 1316)
	<<***extension***>> *(p. 213)* The duration a reader ignores consecutively received heartbeats
DDS_RtpsReliableReaderProtocol_t (p. 1314)	**nack_period** (p. 1316)
	<<***extension***>> *(p. 213)* The period at which to send NACKs
DDS_RtpsReliableReaderProtocol_t (p. 1314)	**receive_window_size** (p. 1317)
	<<***extension***>> *(p. 213)* The number of received out-of-order samples a reader can keep at a time
DDS_RtpsReliableReaderProtocol_t (p. 1314)	**round_trip_time** (p. 1317)
	<<***extension***>> *(p. 213)* The duration from sending a NACK to receiving a repair of a sample
DDS_RtpsReliableReaderProtocol_t (p. 1314)	**app_ack_period** (p. 1318)
	<<***extension***>> *(p. 213)* The period at which application-level acknowledgment messages are sent
DDS_RtpsReliableReaderProtocol_t (p. 1314)	**min_app_ack_response_keep_duration** (p. 1318)
	<<***extension***>> *(p. 213)* Minimum duration for which application-level acknowledgment response data is kept
DDS_RtpsReliableReaderProtocol_t (p. 1314)	**samples_per_app_ack** (p. 1319)
	<<***extension***>> *(p. 213)* The minimum number of samples acknowledged by one application-level acknowledgment message
DDS_RtpsReliableWriterProtocol_t (p. 1320)	**low_watermark** (p. 1321)

	<<***extension***>> *(*p. *213)* When the number of unacknowledged samples in the current send window of a reliable writer meets or falls below this threshold, the **DDS_RELIABLE_WRITER_CACHE_CHANGED↩ _STATUS** (p. 545) is considered to have changed.
DDS_RtpsReliableWriterProtocol_t (p. 1320)	**high_watermark** (p. 1322)
	<<***extension***>> *(*p. *213)* When the number of unacknowledged samples in the current send window of a reliable writer meets or exceeds this threshold, the **DDS_RELIABLE_WRITER_CACHE_CHANGED↩ _STATUS** (p. 545) is considered to have changed.
DDS_RtpsReliableWriterProtocol_t (p. 1320)	**virtual_heartbeat_period** (p. 1325)
	<<***extension***>> *(*p. *213)* The period at which to send virtual heartbeats. Virtual heartbeats inform the reliable reader about the range of samples currently present, for each virtual GUID, in the reliable writer's queue.
DDS_RtpsReliableWriterProtocol_t (p. 1320)	**samples_per_virtual_heartbeat** (p. 1325)
	<<***extension***>> *(*p. *213)* The number of samples that a reliable writer has to publish before sending a virtual heartbeat.
DDS_RtpsReliableWriterProtocol_t (p. 1320)	**max_heartbeat_retries** (p. 1326)
	<<***extension***>> *(*p. *213)* The maximum number of *periodic* heartbeat retries before marking a remote reader as inactive.
DDS_RtpsReliableWriterProtocol_t (p. 1320)	**inactivate_nonprogressing_readers** (p. 1326)
	<<***extension***>> *(*p. *213)* Whether to treat remote readers as inactive when their NACKs do not progress
DDS_RtpsReliableWriterProtocol_t (p. 1320)	**heartbeats_per_max_samples** (p. 1327)
	<<***extension***>> *(*p. *213)* The number of piggyback heartbeats sent per max send window.
DDS_RtpsReliableWriterProtocol_t (p. 1320)	**min_nack_response_delay** (p. 1328)
	<<***extension***>> *(*p. *213)* The minimum delay to respond to a NACK or NACK_FRAG.
DDS_RtpsReliableWriterProtocol_t (p. 1320)	**max_nack_response_delay** (p. 1328)
	<<***extension***>> *(*p. *213)* The maximum delay to respond to a NACK or NACK_FRAG.
DDS_RtpsReliableWriterProtocol_t (p. 1320)	**nack_suppression_duration** (p. 1329)
	<<***extension***>> *(*p. *213)* The duration for ignoring consecutive NACKs/NACK_FRAGs that may trigger redundant repairs.
DDS_RtpsReliableWriterProtocol_t (p. 1320)	**max_bytes_per_nack_response** (p. 1329)

	<<***extension***>> *(p. 213)* The maximum total message size when resending rejected samples or data fragments.
DDS_RtpsReliableWriterProtocol_t (p. 1320)	**disable_positive_acks_enable_adaptive_sample_keep_duration** (p. 1331)
	[DEPRECATED] (p. 384) <<***extension***>> *(p. 213)* Enables dynamic adjustment of sample keep duration in response to congestion.
DDS_RtpsReliableWriterProtocol_t (p. 1320)	**disable_positive_acks_min_sample_keep_duration** (p. 1330)
	<<***extension***>> *(p. 213)* The minimum duration a sample is queued for ACK-disabled readers.
DDS_RtpsReliableWriterProtocol_t (p. 1320)	**disable_positive_acks_max_sample_keep_duration** (p. 1331)
	<<***extension***>> *(p. 213)* The maximum duration a sample is queued for ACK-disabled readers.
DDS_RtpsReliableWriterProtocol_t (p. 1320)	**disable_positive_acks_decrease_sample_keep_duration_factor** (p. 1332)
	[DEPRECATED] (p. 384) <<***extension***>> *(p. 213)* Controls rate of contraction of dynamic sample keep duration.
DDS_RtpsReliableWriterProtocol_t (p. 1320)	**disable_positive_acks_increase_sample_keep_duration_factor** (p. 1332)
	[DEPRECATED] (p. 384) <<***extension***>> *(p. 213)* Controls rate of growth of dynamic sample keep duration.
DDS_RtpsReliableWriterProtocol_t (p. 1320)	**send_window_update_period** (p. 1334)
	<<***extension***>> *(p. 213)* Period in which send window may be dynamically changed
DDS_RtpsReliableWriterProtocol_t (p. 1320)	**send_window_increase_factor** (p. 1335)
	<<***extension***>> *(p. 213)* Increases send window size by this percentage when reacting dynamically to network conditions
DDS_RtpsReliableWriterProtocol_t (p. 1320)	**send_window_decrease_factor** (p. 1336)
	<<***extension***>> *(p. 213)* Decreases send window size by this percentage when reacting dynamically to network conditions
DDS_RtpsReliableWriterProtocol_t (p. 1320)	**enable_multicast_periodic_heartbeat** (p. 1337)
	<<***extension***>> *(p. 213)* Whether periodic heartbeat messages are sent over multicast
DDS_RtpsReliableWriterProtocol_t (p. 1320)	**multicast_resend_threshold** (p. 1337)

	<<***extension***>> *(*p. *213)* The minimum number of requesting readers needed to trigger a multicast resend
DDS_RtpsReliableWriterProtocol_t (p. 1320)	**disable_repair_piggyback_heartbeat** (p. 1338)
	<<***extension***>> *(*p. *213)* Prevents piggyback heartbeats from being sent with repair samples

7.71 Sample Memory Management QoS

Use these QoS fields to configure how Connext manages memory related to the samples that are stored and managed by the DataReader and DataWriter queues.

Use these QoS fields to configure how Connext manages memory related to the samples that are stored and managed by the DataReader and DataWriter queues.

Basic Settings

DDS_ResourceLimitsQosPolicy (p. 1308)	**max_samples** (p. 1311)
	<<***basic***>> *(*p. *201)* <<***spec***>> *(*p. *209)* Represents the maximum samples the middleware can store for any one **DDSDataWriter** (p. 1589) (or **DDSDataReader** (p. 1556)).
DDS_ResourceLimitsQosPolicy (p. 1308)	**max_instances** (p. 1311)
	<<***basic***>> *(*p. *201)* <<***spec***>> *(*p. *209)* Represents the maximum number of instances a **DDSDataWriter** (p. 1589) (or **DDSData↩ Reader** (p. 1556)) can manage.
DDS_ResourceLimitsQosPolicy (p. 1308)	**initial_samples** (p. 1312)
	<<***basic***>> *(*p. *201)* <<***extension***>> *(*p. *213)* Represents the initial samples the middleware will store for any one **DDSDataWriter** (p. 1589) (or **DDSDataReader** (p. 1556)).
DDS_ResourceLimitsQosPolicy (p. 1308)	**initial_instances** (p. 1312)
	<<***basic***>> *(*p. *201)* <<***extension***>> *(*p. *213)* Represents the initial number of instances a **DDSDataWriter** (p. 1589) (or **DDSData↩ Reader** (p. 1556)) will manage.

Advanced Settings

DDS_PersistentStorageSettings (p. 1242)	**writer_instance_cache_allocation** (p. 1246)
	<<***extension***>> *(*p. *213)* Configures the resource limits associated with the instance durable writer history cache.

DDS_PersistentStorageSettings (p. 1242)	**writer_sample_cache_allocation** (p. 1247)
	<<***extension***>> *(p. 213)* Configures the resource limits associated with the sample durable writer history cache.
DDS_DurabilityServiceQosPolicy (p. 1009)	**history_kind** (p. 1010)
	<<***spec***>> *(p. 209)* The kind of history to apply in recouping durable data.
DDS_DurabilityServiceQosPolicy (p. 1009)	**history_depth** (p. 1011)
	<<***spec***>> *(p. 209)* Setting to use for the **DDS_DurabilityQosPolicy::writer_depth** (p. 1008) when recouping durable data.
DDS_DurabilityServiceQosPolicy (p. 1009)	**max_samples** (p. 1011)
	<<***spec***>> *(p. 209)* Part of resource limits QoS policy to apply when feeding a late joiner.
DDS_DurabilityServiceQosPolicy (p. 1009)	**max_instances** (p. 1011)
	<<***spec***>> *(p. 209)* Part of resource limits QoS policy to apply when feeding a late joiner.
DDS_DurabilityServiceQosPolicy (p. 1009)	**max_samples_per_instance** (p. 1012)
	<<***spec***>> *(p. 209)* Part of resource limits QoS policy to apply when feeding a late joiner.
DDS_ResourceLimitsQosPolicy (p. 1308)	**max_samples_per_instance** (p. 1311)
	<<***spec***>> *(p. 209)* Represents the maximum number of samples of any one instance a **DDSDataWriter** (p. 1589) (or **DDSDataReader** (p. 1556)) can manage.
DDS_WriterDataLifecycleQosPolicy (p. 1524)	**autodispose_unregistered_instances** (p. 1525)
	<<***spec***>> *(p. 209)* Boolean flag that controls whether or not a writer will also dispose of an instance when the **DDSDataWriter** (p. 1589) unregisters from the instance by means of the unregister operations.
DDS_WriterDataLifecycleQosPolicy (p. 1524)	**autopurge_unregistered_instances_delay** (p. 1525)
	<<***extension***>> *(p. 213)* Maximum duration for which the **DDSDataWriter** (p. 1589) will maintain information regarding an instance once it has unregistered the instance.
DDS_WriterDataLifecycleQosPolicy (p. 1524)	**autopurge_disposed_instances_delay** (p. 1526)
	<<***extension***>> *(p. 213)* Maximum duration for which the **DDSDataWriter** (p. 1589) will maintain information regarding an instance once it has disposed the instance.
DDS_ReaderDataLifecycleQosPolicy (p. 1289)	**autopurge_nowriter_samples_delay** (p. 1291)

	<<***spec***>> *(p. 209)* Minimum duration for which the **DDSDataReader** (p. 1556) will maintain samples for an instance once its `instance_state` becomes **DDS_NOT_ALIVE_NO_WRITERS_INSTANCE_STATE** (p. 166).
DDS_ReaderDataLifecycleQosPolicy (p. 1289)	**autopurge_disposed_samples_delay** (p. 1291)
	<<***spec***>> *(p. 209)* Minimum duration for which the **DDSDataReader** (p. 1556) will maintain samples for an instance once its `instance_state` becomes **DDS_NOT_ALIVE_DISPOSED_INSTANCE_↩ STATE** (p. 166).
DDS_ReaderDataLifecycleQosPolicy (p. 1289)	**autopurge_disposed_instances_delay** (p. 1291)
	<<***extension***>> *(p. 213)* Minimum duration for which the **DDSDataReader** (p. 1556) will maintain information about a received instance once its `instance_state` becomes **DDS_NOT_ALIVE_DISPOSED_INSTANCE↩ _STATE** (p. 166) and there are no samples for the instance in the DataReader queue.
DDS_ReaderDataLifecycleQosPolicy (p. 1289)	**autopurge_nowriter_instances_delay** (p. 1292)
	<<***extension***>> *(p. 213)* Minimum duration for which the **DDSDataReader** (p. 1556) will maintain information about a received instance once its `instance_↩ state` becomes **DDS_NOT_ALIVE_NO_WRITERS↩ _INSTANCE_STATE** (p. 166) and there are no samples for the instance in the DataReader queue.
DDS_DataReaderResourceLimitsQosPolicy (p. 867)	**max_fragmented_samples** (p. 874)
	<<***extension***>> *(p. 213)* The maximum number of samples for which the **DDSDataReader** (p. 1556) may store fragments at a given point in time.
DDS_DataReaderResourceLimitsQosPolicy (p. 867)	**initial_fragmented_samples** (p. 875)
	<<***extension***>> *(p. 213)* The initial number of samples for which a **DDSDataReader** (p. 1556) may store fragments.
DDS_DataReaderResourceLimitsQosPolicy (p. 867)	**max_fragmented_samples_per_remote_writer** (p. 875)
	<<***extension***>> *(p. 213)* The maximum number of samples per remote writer for which a **DDSDataReader** (p. 1556) may store fragments.
DDS_DataReaderResourceLimitsQosPolicy (p. 867)	**max_fragments_per_sample** (p. 876)
	<<***extension***>> *(p. 213)* Maximum number of fragments for a single sample.
DDS_DataReaderResourceLimitsQosPolicy (p. 867)	**dynamically_allocate_fragmented_samples** (p. 876)

	<<***extension***>> *(p. 213)* Determines whether the **DDSDataReader** (p. 1556) pre-allocates storage for storing fragmented samples.
DDS_DataWriterResourceLimitsQosPolicy (p. 917)	**initial_batches** (p. 920)
	<<***extension***>> *(p. 213)* Represents the initial number of batches a **DDSDataWriter** (p. 1589) will manage.
DDS_DataWriterResourceLimitsQosPolicy (p. 917)	**max_batches** (p. 921)
	<<***extension***>> *(p. 213)* Represents the maximum number of batches a **DDSDataWriter** (p. 1589) will manage.
DDS_DataWriterResourceLimitsQosPolicy (p. 917)	**writer_loaned_sample_allocation** (p. 925)
	<<***extension***>> *(p. 213)* Represents the allocation settings of loaned samples managed by a **DDSDataWriter** (p. 1589).
DDS_DataWriterResourceLimitsQosPolicy (p. 917)	**initialize_writer_loaned_sample** (p. 925)
	<<***extension***>> *(p. 213)* Whether or not to initialize loaned samples returned by a **DDSDataWriter** (p. 1589)
DDS_DiscoveryConfigQosPolicy (p. 934)	**participant_reader_resource_limits** (p. 941)
	<<***extension***>> *(p. 213)* Resource limits.
DDS_DiscoveryConfigQosPolicy (p. 934)	**publication_reader_resource_limits** (p. 942)
	<<***extension***>> *(p. 213)* Resource limits.
DDS_DiscoveryConfigQosPolicy (p. 934)	**subscription_reader_resource_limits** (p. 943)
	<<***extension***>> *(p. 213)* Resource limits.
DDS_DiscoveryConfigQosPolicy (p. 934)	**participant_configuration_reader_resource_limits** (p. 959)
	<<***extension***>> *(p. 213)* Resource limits for the built-in topic participant configuration reader.
DDS_BuiltinTopicReaderResourceLimits_t (p. 815)	**initial_samples** (p. 816)
	<<***extension***>> *(p. 213)* Initial number of samples.
DDS_BuiltinTopicReaderResourceLimits_t (p. 815)	**max_samples** (p. 816)
	<<***extension***>> *(p. 213)* Maximum number of samples.

7.72 Sample Ordering

Use these QoS fields to configure how samples are ordered when stored and accessed by a DataReader.

Use these QoS fields to configure how samples are ordered when stored and accessed by a DataReader.

Basic Settings

DDS_DestinationOrderQosPolicy (p. 931)	**kind** (p. 933)
	<<***basic***>> *(p. 201)* <<***spec***>> *(p. 209)* Specifies the desired kind of destination order.

Advanced Settings

DDS_PresentationQosPolicy (p. 1249)	**access_scope** (p. 1252)
	<<***spec***>> *(p. 209)* Determines the largest scope spanning the entities for which the order and coherency of changes can be preserved.
DDS_PresentationQosPolicy (p. 1249)	**coherent_access** (p. 1252)
	<<***spec***>> *(p. 209)* Specifies support for *coherent* access. Controls whether coherent access is supported within the scope `access_scope`.
DDS_PresentationQosPolicy (p. 1249)	**ordered_access** (p. 1252)
	<<***spec***>> *(p. 209)* Specifies support for *ordered* access to the samples received at the subscription end. Controls whether ordered access is supported within the scope `access_scope`.
DDS_PresentationQosPolicy (p. 1249)	**drop_incomplete_coherent_set** (p. 1253)
	<<***extension***>> *(p. 213)* Indicates whether or not a **DDSDataReader** (p. 1556) should drop samples from an incomplete coherent set (one for which not all the samples were received). Such samples are reported as lost in the SAMPLE_LOST Status.
DDS_DestinationOrderQosPolicy (p. 931)	**scope** (p. 933)
	<<***spec***>> *(p. 209)* Specifies the desired scope of the source destination order.
DDS_DestinationOrderQosPolicy (p. 931)	**source_timestamp_tolerance** (p. 933)
	<<***extension***>> *(p. 213)* Allowed tolerance between source timestamps of consecutive samples.

7.73 Security

Use these QoS fields to enable and configure some security-related capabilities in RTI Connext.

Use these QoS fields to enable and configure some security-related capabilities in RTI Connext.

For a complete security solution, you should use the Security Plugins. See the `RTI Security Plugins User's Manual` for more information.

The QoS listed on this page mostly relate to features that are enabled when the Security Plugins are being used.

Basic Settings

DDS_MonitoringLoggingForwardingSettings (p. 1204)	**security_event_forwarding_level** (p. 1204)
	*<<**basic**>> (p. 201) <<**extension**>> (p. 213)* Log messages with **NDDS_CONFIG_LOG_FACILITY_↩ SECURITY_EVENT** (p. 727) and a log level more verbose than this value will not be distributed by RTI Monitoring Library 2.0.

Advanced Settings

DDS_TypeSupportQosPolicy (p. 1499)	**cdr_padding_kind** (p. 1500)
	[DEPRECATED] (p. 384) *<<**extension**>> (p. 213)* Determines whether or not the padding bytes will be set to zero during CDR serialization.
DDS_WireProtocolQosPolicy (p. 1511)	**compute_crc** (p. 1518)
	*<<**extension**>> (p. 213)* Adds RTPS CRC submessage to every message when this field is set to **DDS_BOOLEAN_TRUE** (p. 513).
DDS_WireProtocolQosPolicy (p. 1511)	**check_crc** (p. 1518)
	*<<**extension**>> (p. 213)* Checks if the received RTPS message is valid by comparing the computed CRC with the received RTPS CRC submessage when this field is set to **DDS_BOOLEAN_TRUE** (p. 513).
DDS_DiscoveryConfigQosPolicy (p. 934)	**secure_volatile_writer** (p. 954)
	*<<**extension**>> (p. 213)* RTPS protocol-related configuration settings for the RTPS reliable writer associated with the built-in secure volatile writer.
DDS_DiscoveryConfigQosPolicy (p. 934)	**secure_volatile_writer_publish_mode** (p. 955)
	*<<**extension**>> (p. 213)* Publish mode policy for the built-in secure volatile writer.
DDS_DiscoveryConfigQosPolicy (p. 934)	**secure_volatile_reader** (p. 955)
	*<<**extension**>> (p. 213)* RTPS reliable reader protocol-related configuration settings for the built-in secure volatile reader.
DDS_DataTags (p. 889)	**tags** (p. 889)
	*<<**spec**>> (p. 209)* Sequence of data tags.

7.74 Throughput

Use these QoS fields to configure features which may be helpful to increase throughput.

Use these QoS fields to configure features which may be helpful to increase throughput.

Basic Settings

DDS_BatchQosPolicy (p. 810)	**enable** (p. 810)
	<<***basic***>> *(p. 201)* <<***extension***>> *(p. 213)* Specifies whether or not batching is enabled.
DDS_BatchQosPolicy (p. 810)	**max_data_bytes** (p. 811)
	<<***basic***>> *(p. 201)* <<***extension***>> *(p. 213)* The maximum cumulative length of all serialized samples in a batch.
DDS_BatchQosPolicy (p. 810)	**max_samples** (p. 811)
	<<***basic***>> *(p. 201)* <<***extension***>> *(p. 213)* The maximum number of samples in a batch.

Advanced Settings

DDS_BatchQosPolicy (p. 810)	**max_flush_delay** (p. 812)
	<<***extension***>> *(p. 213)* The maximum flush delay.
DDS_BatchQosPolicy (p. 810)	**source_timestamp_resolution** (p. 812)
	<<***extension***>> *(p. 213)* Batch source timestamp resolution.
DDS_BatchQosPolicy (p. 810)	**thread_safe_write** (p. 813)
	<<***extension***>> *(p. 213)* Determines whether or not the write operation is thread safe.

7.75 Transports

Use these QoS fields to configure aspects of the transports (udp, shared memory, etc.) used for communication.

Use these QoS fields to configure aspects of the transports (udp, shared memory, etc.) used for communication.

Basic Settings

DDS_TransportBuiltinQosPolicy (p. 1408)	**mask** (p. 1408)
	*<<**basic**>>* (p. *201*) *<<**extension**>>* (p. *213*) Specifies the built-in transports that are registered automatically when the **DDSDomainParticipant** (p. 1619) is enabled.
DDS_TransportMulticastQosPolicy (p. 1415)	**value** (p. 1416)
	*<<**basic**>>* (p. *201*) *<<**extension**>>* (p. *213*) A sequence of multicast communications settings.
DDS_DiscoveryQosPolicy (p. 960)	**initial_peers** (p. 962)
	*<<**basic**>>* (p. *201*) *<<**extension**>>* (p. *213*) Determines the initial list of peers that will be contacted by the Discovery mechanism to send announcements about the presence of this participant.
DDS_DiscoveryQosPolicy (p. 960)	**multicast_receive_addresses** (p. 962)
	*<<**basic**>>* (p. *201*) *<<**extension**>>* (p. *213*) Specifies the multicast group addresses on which discovery-related **meta-traffic** can be received by the DomainParticipant.
DDS_DiscoveryQosPolicy (p. 960)	**accept_unknown_peers** (p. 963)
	*<<**basic**>>* (p. *201*) *<<**extension**>>* (p. *213*) Whether to accept a new participant that is not in the initial peers list.

Advanced Settings

DDS_TransportPriorityQosPolicy (p. 1419)	**value** (p. 1421)
	*<<**spec**>>* (p. *209*) This policy is a hint to the infrastructure as to how to set the priority of the underlying transport used to send the data.
DDS_TransportSelectionQosPolicy (p. 1421)	**enabled_transports** (p. 1422)
	*<<**extension**>>* (p. *213*) A sequence of transport aliases that specifies the transport instances available for use by the entity.
DDS_TransportUnicastQosPolicy (p. 1422)	**value** (p. 1424)
	*<<**extension**>>* (p. *213*) A sequence of unicast communication settings.
DDS_TransportMulticastMappingQosPolicy (p. 1413)	**value** (p. 1415)
	*<<**extension**>>* (p. *213*) A sequence of multicast communication mappings.
DDS_TransportMulticastQosPolicy (p. 1415)	**kind** (p. 1416)
	*<<**extension**>>* (p. *213*) A value that specifies a way to determine how to obtain the multicast address.

DDS_DiscoveryQosPolicy (p. 960)	**enabled_transports** (p. 961)
	<<***extension***>> *(p. 213)* The transports available for use by the Discovery mechanism.
DDS_DiscoveryQosPolicy (p. 960)	**metatraffic_transport_priority** (p. 963)
	<<***extension***>> *(p. 213)* The transport priority to use for the Discovery meta-traffic.
DDS_DiscoveryQosPolicy (p. 960)	**enable_endpoint_discovery** (p. 964)
	[DEPRECATED] (p. 384) <<***extension***>> *(p. 213)* Whether to automatically enable endpoint discovery for all the remote participants.
DDS_DomainParticipantResourceLimitsQosPolicy (p. 977)	**transport_info_list_max_length** (p. 1001)
	<<***extension***>> *(p. 213)* Maximum number of installed transports to send and receive information about in **DDS_ParticipantBuiltinTopicData::transport_info** (p. 1232).
DDS_DiscoveryConfigQosPolicy (p. 934)	**locator_reachability_assert_period** (p. 952)
	<<***extension***>> *(p. 213)* Period at which this DomainParticipant will assert the locators discovered from other DomainParticipants.
DDS_DiscoveryConfigQosPolicy (p. 934)	**locator_reachability_change_detection_period** (p. 953)
	<<***extension***>> *(p. 213)* Period at which this DomainParticipant will check if its locators are reachable from other DomainParticipants.
DDS_DiscoveryConfigQosPolicy (p. 934)	**dns_tracker_polling_period** (p. 956)
	<<***extension***>> *(p. 213)* Duration that specifies the period used by the DNS tracker to poll the DNS service and check for changes in the hostnames.
DDS_RtpsReliableWriterProtocol_t (p. 1320)	**enable_multicast_periodic_heartbeat** (p. 1337)
	<<***extension***>> *(p. 213)* Whether periodic heartbeat messages are sent over multicast
DDS_RtpsReliableWriterProtocol_t (p. 1320)	**multicast_resend_threshold** (p. 1337)
	<<***extension***>> *(p. 213)* The minimum number of requesting readers needed to trigger a multicast resend
DDS_ChannelSettings_t (p. 822)	**multicast_settings** (p. 822)
	<<***extension***>> *(p. 213)* A sequence of **DDS_↩ TransportMulticastSettings_t** (p. 1417) used to configure the multicast addresses associated with a channel.

DDS_ChannelSettings_t (p. 822)	**priority** (p. 823)
	<<***extension***>> *(p. 213)* Publication priority

7.76 Topic Queries

Use these QoS fields to configure the topic query feature.

Use these QoS fields to configure the topic query feature.

See `Querying Data`, in the *Core Libraries User's Manual*, for more information about this feature.

Basic Settings

DDS_TopicQueryDispatchQosPolicy (p. 1404)	**publication_period** (p. 1405)
	<<***basic***>> *(p. 201)* <<***extension***>> *(p. 213)* Sets the periodic interval at which samples are published.

Advanced Settings

DDS_AsynchronousPublisherQosPolicy (p. 798)	**disable_topic_query_publication** (p. 801)
	[DEPRECATED] (p. **384**) <<***extension***>> *(p. 213)* Disable topic query publication.
DDS_AsynchronousPublisherQosPolicy (p. 798)	**topic_query_publication_thread** (p. 802)
	<<***extension***>> *(p. 213)* Settings of the **DDSTopicQuery** (p. 1895) publication thread.
DDS_DataReaderResourceLimitsQosPolicy (p. 867)	**initial_topic_queries** (p. 882)
	<<***extension***>> *(p. 213)* The initial number of TopicQueries allocated by a **DDSDataReader** (p. 1556).
DDS_DataReaderResourceLimitsQosPolicy (p. 867)	**max_topic_queries** (p. 882)
	<<***extension***>> *(p. 213)* The maximum number of active TopicQueries that a **DDSDataReader** (p. 1556) can create.
DDS_DataWriterResourceLimitsQosPolicy (p. 917)	**initial_active_topic_queries** (p. 924)
	<<***extension***>> *(p. 213)* Represents the initial number of active topic queries a **DDSDataWriter** (p. 1589) will manage.
DDS_DataWriterResourceLimitsQosPolicy (p. 917)	**max_active_topic_queries** (p. 924)

	<<***extension***>> *(p. 213)* Represents the maximum number of active topic queries a **DDSDataWriter** (p. 1589) will manage.
DDS_DomainParticipantResourceLimitsQosPolicy (p. 977)	**remote_topic_query_allocation** (p. 1001)
	<<***extension***>> *(p. 213)* Allocation settings applied to remote TopicQueries.
DDS_DomainParticipantResourceLimitsQosPolicy (p. 977)	**remote_topic_query_hash_buckets** (p. 1002)
	[DEPRECATED] (p. **384**) <<***extension***>> *(p. 213)* Number of hash buckets for remote TopicQueries.
DDS_TopicQueryDispatchQosPolicy (p. 1404)	**enable** (p. 1405)
	<<***extension***>> *(p. 213)* Allows this writer to dispatch TopicQueries
DDS_TopicQueryDispatchQosPolicy (p. 1404)	**samples_per_period** (p. 1405)
	<<***extension***>> *(p. 213)* Sets the maximum number of samples to publish in each publication_period.

7.77 Type System

Use these QoS fields to configure aspects of the type system.

Use these QoS fields to configure aspects of the type system.

These settings include how types are sent during discovery, what criteria is used when types should be matched, and how data should be serialized.

Advanced Settings

DDS_DataRepresentationQosPolicy (p. 886)	**value** (p. 888)
	<<***spec***>> *(p. 209)* Sequence of representation identifiers.
DDS_TypeConsistencyEnforcementQosPolicy (p. 1493)	**kind** (p. 1494)
	<<***spec***>> *(p. 209)* Type consistency kind.
DDS_TypeConsistencyEnforcementQosPolicy (p. 1493)	**ignore_sequence_bounds** (p. 1495)
	<<***spec***>> *(p. 209)* Controls whether sequence bounds are taken into consideration for type assignability.

DDS_TypeConsistencyEnforcementQosPolicy (p. 1493)	**ignore_string_bounds** (p. 1495)
	<<***spec***>> *(p. 209)* Controls whether string bounds are taken into consideration for type assignability.
DDS_TypeConsistencyEnforcementQosPolicy (p. 1493)	**ignore_member_names** (p. 1495)
	<<***spec***>> *(p. 209)* Controls whether member names are taken into consideration for type assignability.
DDS_TypeConsistencyEnforcementQosPolicy (p. 1493)	**prevent_type_widening** (p. 1496)
	<<***spec***>> *(p. 209)* Controls whether type widening is allowed.
DDS_TypeConsistencyEnforcementQosPolicy (p. 1493)	**force_type_validation** (p. 1496)
	<<***spec***>> *(p. 209)* Controls whether type information must be available in order to complete matching between a **DDSDataWriter** (p. 1589) and a **DDSData↩ Reader** (p. 1556).
DDS_TypeConsistencyEnforcementQosPolicy (p. 1493)	**ignore_enum_literal_names** (p. 1496)
	<<***spec***>> *(p. 209)* Controls whether enumeration constant names are taken into consideration for type assignability.
DDS_DomainParticipantResourceLimitsQosPolicy (p. 977)	**type_code_max_serialized_length** (p. 994)
	[DEPRECATED] (p. 384) <<***extension***>> *(p. 213)* Maximum size of serialized string for type code.
DDS_DomainParticipantResourceLimitsQosPolicy (p. 977)	**type_object_max_serialized_length** (p. 994)
	<<***extension***>> *(p. 213)* The maximum length, in bytes, that the buffer to serialize a TypeObject v1 can consume.
DDS_DomainParticipantResourceLimitsQosPolicy (p. 977)	**type_object_max_deserialized_length** (p. 996)
	<<***extension***>> *(p. 213)* The maximum number of bytes that a deserialized TypeObject v1 can consume.
DDS_DomainParticipantResourceLimitsQosPolicy (p. 977)	**deserialized_type_object_dynamic_allocation_↩ threshold** (p. 996)
	<<***extension***>> *(p. 213)* A threshold, in bytes, for dynamic memory allocation for the deserialized TypeObject v1.

DDS_DomainParticipantResourceLimitsQosPolicy (p. 977)	**serialized_type_object_dynamic_allocation_threshold** (p. 995)
	<<***extension***>> *(p. 213)* A threshold, in bytes, for dynamic memory allocation for the serialized TypeObject v1.
DDS_DiscoveryConfigQosPolicy (p. 934)	**endpoint_type_object_lb_serialization_threshold** (p. 956)
	<<***extension***>> *(p. 213)* Controls when/if the TypeObject v1 is compressed when propagated in Simple Endpoint Discovery.

7.78 QoS Mutability

Some QoS are mutable after an entity is enabled, and others are not.

Modules

- **Immutable QoS**

  This page lists all QoS that are immutable after an entity has been enabled.
- **Mutable QoS**

  This page lists all QoS that are always mutable.

7.78.1 Detailed Description

Some QoS are mutable after an entity is enabled, and others are not.

7.79 Immutable QoS

This page lists all QoS that are immutable after an entity has been enabled.

This page lists all QoS that are immutable after an entity has been enabled.

QoS fall into one of three mutability categories:

- Always mutable, wheter an entity is enabled or not (see **Mutable QoS** (p. 369)).
- Immutable after entity creation.
- *Mutable* after creation but *immutable* after an entity is enabled.

The QoS that fall into the second two categories are listed on this page.

See also

Changeable (p. 556)

Basic Settings

DDS_DurabilityQosPolicy (p. 1005)	**kind** (p. 1007)
	*<<**basic**>> (p. 201) <<**spec**>> (p. 209)* The kind of durability.
DDS_DurabilityQosPolicy (p. 1005)	**writer_depth** (p. 1008)
	*<<**basic**>> (p. 201) <<**extension**>> (p. 213)* Indicates the number of samples per instance that a durable **DDSDataWriter** (p. 1589) will send to a late-joining **DDSDataReader** (p. 1556).
DDS_DataRepresentationQosPolicy (p. 886)	**compression_settings** (p. 888)
	*<<**basic**>> (p. 201) <<**extension**>> (p. 213)* Structure that contains the compression settings.
DDS_OwnershipQosPolicy (p. 1222)	**kind** (p. 1227)
	*<<**basic**>> (p. 201) <<**spec**>> (p. 209)* The kind of ownership
DDS_LivelinessQosPolicy (p. 1180)	**kind** (p. 1182)
	*<<**basic**>> (p. 201) <<**spec**>> (p. 209)* The kind of liveliness desired.
DDS_LivelinessQosPolicy (p. 1180)	**lease_duration** (p. 1182)
	*<<**basic**>> (p. 201) <<**spec**>> (p. 209)* The duration within which a **DDSDataWriter** (p. 1589) must be asserted, or else it is assumed to be not alive.
DDS_LivelinessQosPolicy (p. 1180)	**assertions_per_lease_duration** (p. 1183)
	*<<**basic**>> (p. 201) <<**extension**>> (p. 213)* The number of assertions a **DDSDataWriter** (p. 1589) will send during a its **DDS_LivelinessQosPolicy::lease_duration** (p. 1182).
DDS_ReliabilityQosPolicy (p. 1295)	**kind** (p. 1298)
	*<<**basic**>> (p. 201) <<**spec**>> (p. 209)* Kind of reliability.
DDS_ReliabilityQosPolicy (p. 1295)	**max_blocking_time** (p. 1298)
	*<<**basic**>> (p. 201) <<**spec**>> (p. 209)* The maximum time a DataWriter can block on a write() call.
DDS_DestinationOrderQosPolicy (p. 931)	**kind** (p. 933)
	*<<**basic**>> (p. 201) <<**spec**>> (p. 209)* Specifies the desired kind of destination order.
DDS_HistoryQosPolicy (p. 1162)	**kind** (p. 1164)
	*<<**basic**>> (p. 201) <<**spec**>> (p. 209)* Specifies the kind of history to be kept.
DDS_HistoryQosPolicy (p. 1162)	**depth** (p. 1164)

	*<<**basic**>> (p. 201) <<**spec**>> (p. 209)* Specifies the number of samples per instance to be kept, when the `kind` is **DDS_KEEP_LAST_HISTORY_QOS** (p. 604)
DDS_ResourceLimitsQosPolicy (p. 1308)	**max_samples** (p. 1311)
	*<<**basic**>> (p. 201) <<**spec**>> (p. 209)* Represents the maximum samples the middleware can store for any one **DDSDataWriter** (p. 1589) (or **DDSDataReader** (p. 1556)).
DDS_ResourceLimitsQosPolicy (p. 1308)	**max_instances** (p. 1311)
	*<<**basic**>> (p. 201) <<**spec**>> (p. 209)* Represents the maximum number of instances a **DDSDataWriter** (p. 1589) (or **DDSDataReader** (p. 1556)) can manage.
DDS_ResourceLimitsQosPolicy (p. 1308)	**initial_samples** (p. 1312)
	*<<**basic**>> (p. 201) <<**extension**>> (p. 213)* Represents the initial samples the middleware will store for any one **DDSDataWriter** (p. 1589) (or **DDSDataReader** (p. 1556)).
DDS_ResourceLimitsQosPolicy (p. 1308)	**initial_instances** (p. 1312)
	*<<**basic**>> (p. 201) <<**extension**>> (p. 213)* Represents the initial number of instances a **DDSDataWriter** (p. 1589) (or **DDSDataReader** (p. 1556)) will manage.
DDS_PublishModeQosPolicy (p. 1279)	**kind** (p. 1281)
	*<<**basic**>> (p. 201) <<**extension**>> (p. 213)* Publishing mode
DDS_WireProtocolQosPolicy (p. 1511)	**participant_id** (p. 1514)
	*<<**basic**>> (p. 201) <<**extension**>> (p. 213)* A value used to distinguish among different participants belonging to the same domain on the same host.
DDS_TransportBuiltinQosPolicy (p. 1408)	**mask** (p. 1408)
	*<<**basic**>> (p. 201) <<**extension**>> (p. 213)* Specifies the built-in transports that are registered automatically when the **DDSDomainParticipant** (p. 1619) is enabled.
DDS_TransportMulticastQosPolicy (p. 1415)	**value** (p. 1416)
	*<<**basic**>> (p. 201) <<**extension**>> (p. 213)* A sequence of multicast communications settings.
DDS_DiscoveryQosPolicy (p. 960)	**initial_peers** (p. 962)
	*<<**basic**>> (p. 201) <<**extension**>> (p. 213)* Determines the initial list of peers that will be contacted by the Discovery mechanism to send announcements about the presence of this participant.
DDS_DiscoveryQosPolicy (p. 960)	**multicast_receive_addresses** (p. 962)

	*<<**basic**>> (p. 201) <<**extension**>> (p. 213)* Specifies the multicast group addresses on which discovery-related **meta-traffic** can be received by the DomainParticipant.
DDS_DiscoveryQosPolicy (p. 960)	**accept_unknown_peers** (p. 963)
	*<<**basic**>> (p. 201) <<**extension**>> (p. 213)* Whether to accept a new participant that is not in the initial peers list.
DDS_DiscoveryConfigQosPolicy (p. 934)	**participant_liveliness_lease_duration** (p. 937)
	*<<**basic**>> (p. 201) <<**extension**>> (p. 213)* The liveliness lease duration for the participant.
DDS_DiscoveryConfigQosPolicy (p. 934)	**participant_liveliness_assert_period** (p. 937)
	*<<**basic**>> (p. 201) <<**extension**>> (p. 213)* The period to assert liveliness for the participant.
DDS_DiscoveryConfigQosPolicy (p. 934)	**participant_announcement_period** (p. 937)
	*<<**basic**>> (p. 201) <<**extension**>> (p. 213)* The period at which a participant announces itself to potential peers when using the Simple Participant Discovery Protocol 2.0 (SPDP2).
DDS_DiscoveryConfigQosPolicy (p. 934)	**initial_participant_announcements** (p. 939)
	*<<**basic**>> (p. 201) <<**extension**>> (p. 213)* The number of initial announcements sent when a participant is first enabled.
DDS_DiscoveryConfigQosPolicy (p. 934)	**new_remote_participant_announcements** (p. 939)
	*<<**basic**>> (p. 201) <<**extension**>> (p. 213)* The number of participant announcements sent when a remote participant is newly discovered.
DDS_DiscoveryConfigQosPolicy (p. 934)	**min_initial_participant_announcement_period** (p. 940)
	*<<**basic**>> (p. 201) <<**extension**>> (p. 213)* The minimum period between initial announcements when a participant is first enabled or when a remote participant is newly discovered.
DDS_DiscoveryConfigQosPolicy (p. 934)	**max_initial_participant_announcement_period** (p. 940)
	*<<**basic**>> (p. 201) <<**extension**>> (p. 213)* The maximum period between initial announcements when a participant is first enabled or when a remote participant is newly discovered.
DDS_RtpsReliableWriterProtocol_t (p. 1320)	**heartbeat_period** (p. 1323)
	*<<**basic**>> (p. 201) <<**extension**>> (p. 213)* The period at which to send heartbeats.
DDS_RtpsReliableWriterProtocol_t (p. 1320)	**fast_heartbeat_period** (p. 1323)
	*<<**basic**>> (p. 201) <<**extension**>> (p. 213)* An alternative heartbeat period used when a reliable writer needs to flush its unacknowledged samples more quickly.
DDS_RtpsReliableWriterProtocol_t (p. 1320)	**late_joiner_heartbeat_period** (p. 1324)

	*<<**basic**>> (p. 201) <<**extension**>> (p. 213)* An alternative heartbeat period used when a reliable reader joins late and needs to be caught up on cached samples of a reliable writer more quickly than the normal heartbeat rate.
DDS_RtpsReliableWriterProtocol_t (p. 1320)	**min_send_window_size** (p. 1333)
	*<<**basic**>> (p. 201) <<**extension**>> (p. 213)* Minimum size of send window of unacknowledged samples
DDS_RtpsReliableWriterProtocol_t (p. 1320)	**max_send_window_size** (p. 1333)
	*<<**basic**>> (p. 201) <<**extension**>> (p. 213)* Maximum size of send window of unacknowledged samples
DDS_EntityNameQosPolicy (p. 1147)	**name** (p. 1148)
	*<<**basic**>> (p. 201) <<**extension**>> (p. 213)* The name of the entity.
DDS_BatchQosPolicy (p. 810)	**enable** (p. 810)
	*<<**basic**>> (p. 201) <<**extension**>> (p. 213)* Specifies whether or not batching is enabled.
DDS_BatchQosPolicy (p. 810)	**max_data_bytes** (p. 811)
	*<<**basic**>> (p. 201) <<**extension**>> (p. 213)* The maximum cumulative length of all serialized samples in a batch.
DDS_BatchQosPolicy (p. 810)	**max_samples** (p. 811)
	*<<**basic**>> (p. 201) <<**extension**>> (p. 213)* The maximum number of samples in a batch.
DDS_ProfileQosPolicy (p. 1257)	**string_profile** (p. 1258)
	*<<**basic**>> (p. 201) <<**extension**>> (p. 213)* Sequence of strings containing a XML document to load.
DDS_ProfileQosPolicy (p. 1257)	**url_profile** (p. 1258)
	*<<**basic**>> (p. 201) <<**extension**>> (p. 213)* Sequence of **URL groups** (p. 392) containing a set of XML documents to load.
DDS_ProfileQosPolicy (p. 1257)	**ignore_user_profile** (p. 1258)
	*<<**basic**>> (p. 201) <<**extension**>> (p. 213)* Ignores the file USER_QOS_PROFILES.xml in the current working directory.
DDS_ProfileQosPolicy (p. 1257)	**ignore_environment_profile** (p. 1258)
	*<<**basic**>> (p. 201) <<**extension**>> (p. 213)* Ignores the value of the **NDDS_QOS_PROFILES** (p. 392).
DDS_ProfileQosPolicy (p. 1257)	**ignore_resource_profile** (p. 1259)
	*<<**basic**>> (p. 201) <<**extension**>> (p. 213)* Ignores the file NDDS_QOS_PROFILES.xml.
DDS_TopicQueryDispatchQosPolicy (p. 1404)	**publication_period** (p. 1405)

	<<***basic***>> *(*p. *201)* <<***extension***>> *(*p. *213)* Sets the periodic interval at which samples are published.

Advanced Settings

DDS_AvailabilityQosPolicy (p. 805)	**enable_required_subscriptions** (p. 808)
	<<***extension***>> *(*p. *213)* Enables support for required subscriptions in a **DDSDataWriter** (p. 1589)
DDS_DurabilityQosPolicy (p. 1005)	**direct_communication** (p. 1007)
	<<***extension***>> *(*p. *213)* Indicates whether or not a TRANSIENT or PERSISTENT **DDSDataReader** (p. 1556) should receive samples directly from a TRANSIENT or PERSISTENT **DDSDataWriter** (p. 1589)
DDS_DurabilityQosPolicy (p. 1005)	**storage_settings** (p. 1008)
	<<***extension***>> *(*p. *213)* Configures durable writer history and durable reader state.
DDS_PersistentStorageSettings (p. 1242)	**enable** (p. 1244)
	<<***extension***>> *(*p. *213)* Enables durable writer history in a **DDSDataWriter** (p. 1589) and durable reader state in a **DDSDataReader** (p. 1556).
DDS_PersistentStorageSettings (p. 1242)	**file_name** (p. 1244)
	<<***extension***>> *(*p. *213)* The file name where the durable writer history or durable reader state will be stored.
DDS_PersistentStorageSettings (p. 1242)	**trace_file_name** (p. 1244)
	<<***extension***>> *(*p. *213)* The file name where to store the SQL statements executed when loading and storing the durable writer history or durable reader state.
DDS_PersistentStorageSettings (p. 1242)	**journal_kind** (p. 1245)
	<<***extension***>> *(*p. *213)* Sets the journal mode of the persistent storage.
DDS_PersistentStorageSettings (p. 1242)	**synchronization_kind** (p. 1245)
	<<***extension***>> *(*p. *213)* Sets the level of synchronization with the physical disk.
DDS_PersistentStorageSettings (p. 1242)	**vacuum** (p. 1245)
	<<***extension***>> *(*p. *213)* Sets the auto-vacuum status of the storage.
DDS_PersistentStorageSettings (p. 1242)	**restore** (p. 1246)

	<<***extension***>> *(p. 213)* Indicates if the persisted writer history or reader state must be restored.
DDS_PersistentStorageSettings (p. 1242)	**writer_instance_cache_allocation** (p. 1246)
	<<***extension***>> *(p. 213)* Configures the resource limits associated with the instance durable writer history cache.
DDS_PersistentStorageSettings (p. 1242)	**writer_sample_cache_allocation** (p. 1247)
	<<***extension***>> *(p. 213)* Configures the resource limits associated with the sample durable writer history cache.
DDS_PersistentStorageSettings (p. 1242)	**writer_memory_state** (p. 1248)
	<<***extension***>> *(p. 213)* Determines how much state will be kept in memory by the durable writer history in order to avoid accessing the persistent storage in disk.
DDS_PersistentStorageSettings (p. 1242)	**reader_checkpoint_frequency** (p. 1248)
	<<***extension***>> *(p. 213)* Controls how often the reader state is stored into the database.
DDS_DurabilityServiceQosPolicy (p. 1009)	**service_cleanup_delay** (p. 1010)
	<<***spec***>> *(p. 209)* Controls when the service is able to remove all information regarding a data instances.
DDS_DurabilityServiceQosPolicy (p. 1009)	**history_kind** (p. 1010)
	<<***spec***>> *(p. 209)* The kind of history to apply in recouping durable data.
DDS_DurabilityServiceQosPolicy (p. 1009)	**history_depth** (p. 1011)
	<<***spec***>> *(p. 209)* Setting to use for the **DDS_↩ DurabilityQosPolicy::writer_depth** (p. 1008) when recouping durable data.
DDS_DurabilityServiceQosPolicy (p. 1009)	**max_samples** (p. 1011)
	<<***spec***>> *(p. 209)* Part of resource limits QoS policy to apply when feeding a late joiner.
DDS_DurabilityServiceQosPolicy (p. 1009)	**max_instances** (p. 1011)
	<<***spec***>> *(p. 209)* Part of resource limits QoS policy to apply when feeding a late joiner.
DDS_DurabilityServiceQosPolicy (p. 1009)	**max_samples_per_instance** (p. 1012)
	<<***spec***>> *(p. 209)* Part of resource limits QoS policy to apply when feeding a late joiner.
DDS_CompressionSettings_t (p. 826)	**compression_ids** (p. 826)
	<<***extension***>> *(p. 213)* Mask that represents the compression algorithms enabled.

DDS_CompressionSettings_t (p. 826)	**writer_compression_level** (p. 827)
	<<***extension***>> *(p. 213)* The level of compression to use when compressing data.
DDS_CompressionSettings_t (p. 826)	**writer_compression_threshold** (p. 827)
	<<***extension***>> *(p. 213)* The threshold, in bytes, above which a serialized sample will be eligible to be compressed.
DDS_DataRepresentationQosPolicy (p. 886)	**value** (p. 888)
	<<***spec***>> *(p. 209)* Sequence of representation identifiers.
DDS_PresentationQosPolicy (p. 1249)	**access_scope** (p. 1252)
	<<***spec***>> *(p. 209)* Determines the largest scope spanning the entities for which the order and coherency of changes can be preserved.
DDS_PresentationQosPolicy (p. 1249)	**coherent_access** (p. 1252)
	<<***spec***>> *(p. 209)* Specifies support for *coherent* access. Controls whether coherent access is supported within the scope `access_scope`.
DDS_PresentationQosPolicy (p. 1249)	**ordered_access** (p. 1252)
	<<***spec***>> *(p. 209)* Specifies support for *ordered* access to the samples received at the subscription end. Controls whether ordered access is supported within the scope `access_scope`.
DDS_PresentationQosPolicy (p. 1249)	**drop_incomplete_coherent_set** (p. 1253)
	<<***extension***>> *(p. 213)* Indicates whether or not a **DDSDataReader** (p. 1556) should drop samples from an incomplete coherent set (one for which not all the samples were received). Such samples are reported as lost in the SAMPLE_LOST Status.
DDS_ReliabilityQosPolicy (p. 1295)	**acknowledgment_kind** (p. 1298)
	<<***extension***>> *(p. 213)* Kind of reliable acknowledgment
DDS_ReliabilityQosPolicy (p. 1295)	**instance_state_consistency_kind** (p. 1299)
	<<***extension***>> *(p. 213)* Whether instance state consistency is enabled
DDS_ReliabilityQosPolicy (p. 1295)	**exact_reliability_kind_match** (p. 1299)
	<<***extension***>> *(p. 213)* Whether an exact reliability kind match is required for endpoint matching.
DDS_DestinationOrderQosPolicy (p. 931)	**scope** (p. 933)

	<<***spec***>> *(p. 209)* Specifies the desired scope of the source destination order.
DDS_DestinationOrderQosPolicy (p. 931)	**source_timestamp_tolerance** (p. 933)
	<<***extension***>> *(p. 213)* Allowed tolerance between source timestamps of consecutive samples.
DDS_ResourceLimitsQosPolicy (p. 1308)	**max_samples_per_instance** (p. 1311)
	<<***spec***>> *(p. 209)* Represents the maximum number of samples of any one instance a **DDSDataWriter** (p. 1589) (or **DDSDataReader** (p. 1556)) can manage.
DDS_ResourceLimitsQosPolicy (p. 1308)	**instance_hash_buckets** (p. 1313)
	[DEPRECATED] (p. **384**) <<***extension***>> *(p. 213)* Number of hash buckets for instances.
DDS_TransportPriorityQosPolicy (p. 1419)	**value** (p. 1421)
	<<***spec***>> *(p. 209)* This policy is a hint to the infrastructure as to how to set the priority of the underlying transport used to send the data.
DDS_TypeSupportQosPolicy (p. 1499)	**plugin_data** (p. 1500)
	[DEPRECATED] (p. **384**) <<***extension***>> *(p. 213)* Value to pass into the type plugin's de-/serialization function
DDS_TypeSupportQosPolicy (p. 1499)	**cdr_padding_kind** (p. 1500)
	[DEPRECATED] (p. **384**) <<***extension***>> *(p. 213)* Determines whether or not the padding bytes will be set to zero during CDR serialization.
DDS_PublishModeQosPolicy (p. 1279)	**flow_controller_name** (p. 1281)
	<<***extension***>> *(p. 213)* Name of the associated flow controller
DDS_PublishModeQosPolicy (p. 1279)	**priority** (p. 1282)
	<<***extension***>> *(p. 213)* Publication priority
DDS_AsynchronousPublisherQosPolicy (p. 798)	**disable_asynchronous_write** (p. 800)
	[DEPRECATED] (p. **384**) <<***extension***>> *(p. 213)* Disable asynchronous publishing.
DDS_AsynchronousPublisherQosPolicy (p. 798)	**thread** (p. 800)
	<<***extension***>> *(p. 213)* Settings of the publishing thread.
DDS_AsynchronousPublisherQosPolicy (p. 798)	**disable_asynchronous_batch** (p. 800)
	[DEPRECATED] (p. **384**) <<***extension***>> *(p. 213)* Disable asynchronous batch flushing.
DDS_AsynchronousPublisherQosPolicy (p. 798)	**asynchronous_batch_thread** (p. 801)

	<<***extension***>> *(p. 213)* Settings of the batch flushing thread.
DDS_AsynchronousPublisherQosPolicy (p. 798)	**disable_topic_query_publication** (p. 801)
	[DEPRECATED] (p. **384**) <<***extension***>> *(p. 213)* Disable topic query publication.
DDS_AsynchronousPublisherQosPolicy (p. 798)	**topic_query_publication_thread** (p. 802)
	<<***extension***>> *(p. 213)* Settings of the **DDSTopic↩ Query** (p. 1895) publication thread.
DDS_WriterDataLifecycleQosPolicy (p. 1524)	**autodispose_unregistered_instances** (p. 1525)
	<<***spec***>> *(p. 209)* Boolean flag that controls whether or not a writer will also dispose of an instance when the **DDSDataWriter** (p. 1589) unregisters from the instance by means of the unregister operations.
DDS_WriterDataLifecycleQosPolicy (p. 1524)	**autopurge_unregistered_instances_delay** (p. 1525)
	<<***extension***>> *(p. 213)* Maximum duration for which the **DDSDataWriter** (p. 1589) will maintain information regarding an instance once it has unregistered the instance.
DDS_WriterDataLifecycleQosPolicy (p. 1524)	**autopurge_disposed_instances_delay** (p. 1526)
	<<***extension***>> *(p. 213)* Maximum duration for which the **DDSDataWriter** (p. 1589) will maintain information regarding an instance once it has disposed the instance.
DDS_ReaderDataLifecycleQosPolicy (p. 1289)	**autopurge_nowriter_samples_delay** (p. 1291)
	<<***spec***>> *(p. 209)* Minimum duration for which the **DDSDataReader** (p. 1556) will maintain samples for an instance once its `instance_state` becomes **DDS_NOT_ALIVE_NO_WRITERS_INSTANCE_STATE** (p. 166).
DDS_ReaderDataLifecycleQosPolicy (p. 1289)	**autopurge_disposed_samples_delay** (p. 1291)
	<<***spec***>> *(p. 209)* Minimum duration for which the **DDSDataReader** (p. 1556) will maintain samples for an instance once its `instance_state` becomes **DDS_NOT_ALIVE_DISPOSED_INSTANCE_↩ STATE** (p. 166).
DDS_ReaderDataLifecycleQosPolicy (p. 1289)	**autopurge_disposed_instances_delay** (p. 1291)
	<<***extension***>> *(p. 213)* Minimum duration for which the **DDSDataReader** (p. 1556) will maintain information about a received instance once its `instance_state` becomes **DDS_NOT_ALIVE_DISPOSED_INSTANCE↩ _STATE** (p. 166) and there are no samples for the instance in the DataReader queue.

DDS_ReaderDataLifecycleQosPolicy (p. 1289)	**autopurge_nowriter_instances_delay** (p. 1292)
	<<***extension***>> *(p. 213)* Minimum duration for which the **DDSDataReader** (p. 1556) will maintain information about a received instance once its `instance_state` becomes **DDS_NOT_ALIVE_NO_WRITERS_INSTANCE_STATE** (p. 166) and there are no samples for the instance in the DataReader queue.
DDS_TypeConsistencyEnforcementQosPolicy (p. 1493)	**kind** (p. 1494)
	<<***spec***>> *(p. 209)* Type consistency kind.
DDS_TypeConsistencyEnforcementQosPolicy (p. 1493)	**ignore_sequence_bounds** (p. 1495)
	<<***spec***>> *(p. 209)* Controls whether sequence bounds are taken into consideration for type assignability.
DDS_TypeConsistencyEnforcementQosPolicy (p. 1493)	**ignore_string_bounds** (p. 1495)
	<<***spec***>> *(p. 209)* Controls whether string bounds are taken into consideration for type assignability.
DDS_TypeConsistencyEnforcementQosPolicy (p. 1493)	**ignore_member_names** (p. 1495)
	<<***spec***>> *(p. 209)* Controls whether member names are taken into consideration for type assignability.
DDS_TypeConsistencyEnforcementQosPolicy (p. 1493)	**prevent_type_widening** (p. 1496)
	<<***spec***>> *(p. 209)* Controls whether type widening is allowed.
DDS_TypeConsistencyEnforcementQosPolicy (p. 1493)	**force_type_validation** (p. 1496)
	<<***spec***>> *(p. 209)* Controls whether type information must be available in order to complete matching between a **DDSDataWriter** (p. 1589) and a **DDSDataReader** (p. 1556).
DDS_TypeConsistencyEnforcementQosPolicy (p. 1493)	**ignore_enum_literal_names** (p. 1496)
	<<***spec***>> *(p. 209)* Controls whether enumeration constant names are taken into consideration for type assignability.
DDS_WireProtocolQosPolicy (p. 1511)	**rtps_host_id** (p. 1515)
	<<***extension***>> *(p. 213)* The RTPS Host ID of the domain participant.
DDS_WireProtocolQosPolicy (p. 1511)	**rtps_app_id** (p. 1516)

	<<***extension***>> *(*p. *213)* The RTPS App ID of the domain participant.
DDS_WireProtocolQosPolicy (p. 1511)	**rtps_instance_id** (p. 1516)
	<<***extension***>> *(*p. *213)* The RTPS Instance ID of the **DDSDomainParticipant** (p. 1619).
DDS_WireProtocolQosPolicy (p. 1511)	**rtps_well_known_ports** (p. 1517)
	<<***extension***>> *(*p. *213)* Configures the RTPS well-known port mappings.
DDS_WireProtocolQosPolicy (p. 1511)	**rtps_reserved_port_mask** (p. 1517)
	<<***extension***>> *(*p. *213)* Specifies which well-known ports to reserve when enabling the participant.
DDS_WireProtocolQosPolicy (p. 1511)	**rtps_auto_id_kind** (p. 1518)
	<<***extension***>> *(*p. *213)* Kind of auto mechanism used to calculate the GUID prefix.
DDS_WireProtocolQosPolicy (p. 1511)	**compute_crc** (p. 1518)
	<<***extension***>> *(*p. *213)* Adds RTPS CRC submessage to every message when this field is set to **DDS_↩ BOOLEAN_TRUE** (p. 513).
DDS_WireProtocolQosPolicy (p. 1511)	**check_crc** (p. 1518)
	<<***extension***>> *(*p. *213)* Checks if the received RTPS message is valid by comparing the computed CRC with the received RTPS CRC submessage when this field is set to **DDS_BOOLEAN_TRUE** (p. 513).
DDS_TransportSelectionQosPolicy (p. 1421)	**enabled_transports** (p. 1422)
	<<***extension***>> *(*p. *213)* A sequence of transport aliases that specifies the transport instances available for use by the entity.
DDS_TransportUnicastQosPolicy (p. 1422)	**value** (p. 1424)
	<<***extension***>> *(*p. *213)* A sequence of unicast communication settings.
DDS_TransportMulticastMappingQosPolicy (p. 1413)	**value** (p. 1415)
	<<***extension***>> *(*p. *213)* A sequence of multicast communication mappings.
DDS_TransportMulticastQosPolicy (p. 1415)	**kind** (p. 1416)
	<<***extension***>> *(*p. *213)* A value that specifies a way to determine how to obtain the multicast address.
DDS_DiscoveryQosPolicy (p. 960)	**enabled_transports** (p. 961)

	<<***extension***>> *(p. 213)* The transports available for use by the Discovery mechanism.
DDS_DiscoveryQosPolicy (p. 960)	**metatraffic_transport_priority** (p. 963)
	<<***extension***>> *(p. 213)* The transport priority to use for the Discovery meta-traffic.
DDS_DiscoveryQosPolicy (p. 960)	**enable_endpoint_discovery** (p. 964)
	[DEPRECATED] (p. 384) <<***extension***>> *(p. 213)* Whether to automatically enable endpoint discovery for all the remote participants.
DDS_DataReaderResourceLimitsQosPolicy (p. 867)	**max_remote_writers** (p. 870)
	<<***extension***>> *(p. 213)* The maximum number of remote writers from which a **DDSDataReader** (p. 1556) may read, including all instances.
DDS_DataReaderResourceLimitsQosPolicy (p. 867)	**max_remote_writers_per_instance** (p. 870)
	<<***extension***>> *(p. 213)* The maximum number of remote writers from which a **DDSDataReader** (p. 1556) may read a single instance.
DDS_DataReaderResourceLimitsQosPolicy (p. 867)	**max_samples_per_remote_writer** (p. 870)
	<<***extension***>> *(p. 213)* The maximum number of out-of-order samples from a given remote **DDSData↩ Writer** (p. 1589) that a **DDSDataReader** (p. 1556) may store when maintaining a reliable connection to the **DDSDataWriter** (p. 1589).
DDS_DataReaderResourceLimitsQosPolicy (p. 867)	**max_infos** (p. 871)
	<<***extension***>> *(p. 213)* The maximum number of info units that a **DDSDataReader** (p. 1556) can use to store **DDS_SampleInfo** (p. 1344).
DDS_DataReaderResourceLimitsQosPolicy (p. 867)	**initial_remote_writers** (p. 871)
	<<***extension***>> *(p. 213)* The initial number of remote writers from which a **DDSDataReader** (p. 1556) may read, including all instances.
DDS_DataReaderResourceLimitsQosPolicy (p. 867)	**initial_remote_writers_per_instance** (p. 872)
	<<***extension***>> *(p. 213)* The initial number of remote writers from which a **DDSDataReader** (p. 1556) may read a single instance.
DDS_DataReaderResourceLimitsQosPolicy (p. 867)	**initial_infos** (p. 872)
	<<***extension***>> *(p. 213)* The initial number of info units that a **DDSDataReader** (p. 1556) can have, which are used to store **DDS_SampleInfo** (p. 1344).
DDS_DataReaderResourceLimitsQosPolicy (p. 867)	**initial_outstanding_reads** (p. 873)

	<<***extension***>> *(p. 213)* The initial number of outstanding calls to read/take (or one of their variants) on the same **DDSDataReader** (p. 1556) for which memory has not been returned by calling **FooDataReader::return_loan** (p. 1940).
DDS_DataReaderResourceLimitsQosPolicy (p. 867)	**max_outstanding_reads** (p. 873)
	<<***extension***>> *(p. 213)* The maximum number of outstanding read/take calls (or one of their variants) on the same **DDSDataReader** (p. 1556) for which memory has not been returned by calling **FooDataReader::return_loan** (p. 1940).
DDS_DataReaderResourceLimitsQosPolicy (p. 867)	**max_samples_per_read** (p. 873)
	<<***extension***>> *(p. 213)* The maximum number of data samples that the application can receive from the middleware in a single call to **FooDataReader::read** (p. 1919) or **FooDataReader::take** (p. 1920). If more data exists in the middleware, the application will need to issue multiple read/take calls.
DDS_DataReaderResourceLimitsQosPolicy (p. 867)	**disable_fragmentation_support** (p. 874)
	[DEPRECATED] (p. 384) <<***extension***>> *(p. 213)* Determines whether the **DDSDataReader** (p. 1556) can receive fragmented samples.
DDS_DataReaderResourceLimitsQosPolicy (p. 867)	**max_fragmented_samples** (p. 874)
	<<***extension***>> *(p. 213)* The maximum number of samples for which the **DDSDataReader** (p. 1556) may store fragments at a given point in time.
DDS_DataReaderResourceLimitsQosPolicy (p. 867)	**initial_fragmented_samples** (p. 875)
	<<***extension***>> *(p. 213)* The initial number of samples for which a **DDSDataReader** (p. 1556) may store fragments.
DDS_DataReaderResourceLimitsQosPolicy (p. 867)	**max_fragmented_samples_per_remote_writer** (p. 875)
	<<***extension***>> *(p. 213)* The maximum number of samples per remote writer for which a **DDSDataReader** (p. 1556) may store fragments.
DDS_DataReaderResourceLimitsQosPolicy (p. 867)	**max_fragments_per_sample** (p. 876)
	<<***extension***>> *(p. 213)* Maximum number of fragments for a single sample.
DDS_DataReaderResourceLimitsQosPolicy (p. 867)	**dynamically_allocate_fragmented_samples** (p. 876)
	<<***extension***>> *(p. 213)* Determines whether the **DDSDataReader** (p. 1556) pre-allocates storage for storing fragmented samples.

DDS_DataReaderResourceLimitsQosPolicy (p. 867)	**max_total_instances** (p. 877)
	<<***extension***>> *(p. 213)* Maximum number of instances for which a DataReader will keep state.
DDS_DataReaderResourceLimitsQosPolicy (p. 867)	**max_remote_virtual_writers** (p. 878)
	<<***extension***>> *(p. 213)* The maximum number of remote virtual writers from which a **DDSDataReader** (p. 1556) may read, including all instances.
DDS_DataReaderResourceLimitsQosPolicy (p. 867)	**initial_remote_virtual_writers** (p. 878)
	<<***extension***>> *(p. 213)* The initial number of remote virtual writers from which a **DDSDataReader** (p. 1556) may read, including all instances.
DDS_DataReaderResourceLimitsQosPolicy (p. 867)	**max_remote_virtual_writers_per_instance** (p. 879)
	<<***extension***>> *(p. 213)* The maximum number of virtual remote writers that can be associated with an instance.
DDS_DataReaderResourceLimitsQosPolicy (p. 867)	**initial_remote_virtual_writers_per_instance** (p. 879)
	<<***extension***>> *(p. 213)* The initial number of virtual remote writers per instance.
DDS_DataReaderResourceLimitsQosPolicy (p. 867)	**max_remote_writers_per_sample** (p. 880)
	<<***extension***>> *(p. 213)* The maximum number of remote writers allowed to write the same sample.
DDS_DataReaderResourceLimitsQosPolicy (p. 867)	**max_query_condition_filters** (p. 880)
	<<***extension***>> *(p. 213)* The maximum number of query condition filters a reader is allowed.
DDS_DataReaderResourceLimitsQosPolicy (p. 867)	**max_app_ack_response_length** (p. 881)
	<<***extension***>> *(p. 213)* Maximum length of application-level acknowledgment response data
DDS_DataReaderResourceLimitsQosPolicy (p. 867)	**keep_minimum_state_for_instances** (p. 881)
	<<***extension***>> *(p. 213)* Whether or not keep a minimum instance state for up to **DDS_DataReader↩ ResourceLimitsQosPolicy::max_total_instances** (p. 877).
DDS_DataReaderResourceLimitsQosPolicy (p. 867)	**initial_topic_queries** (p. 882)
	<<***extension***>> *(p. 213)* The initial number of Topic↩ Queries allocated by a **DDSDataReader** (p. 1556).
DDS_DataReaderResourceLimitsQosPolicy (p. 867)	**max_topic_queries** (p. 882)
	<<***extension***>> *(p. 213)* The maximum number of active TopicQueries that a **DDSDataReader** (p. 1556) can create.

DDS_DataReaderResourceLimitsQosPolicy (p. 867)	**shmem_ref_transfer_mode_attached_segment_↩ allocation** (p. 883)
	*<<**extension**>>* *(p. 213)* Allocation resource for the shared memory segments attached by the **DDSData↩ Reader** (p. 1556).
DDS_DataReaderResourceLimitsQosPolicy (p. 867)	**instance_replacement** (p. 883)
	*<<**extension**>>* *(p. 213)* Sets the kind of instances allowed to be replaced for each instance state (**DDS↩ _InstanceStateKind** (p. 165)) when a DataReader reaches **DDS_ResourceLimitsQosPolicy::max_↩ instances** (p. 1311).
DDS_DataReaderResourceLimitsQosPolicy (p. 867)	**autopurge_remote_not_alive_writer_delay** (p. 884)
	*<<**extension**>>* *(p. 213)* Maximum duration for which the **DDSDataReader** (p. 1556) will maintain information regarding a **DDSDataWriter** (p. 1589) once the **DDSDataWriter** (p. 1589) has become not alive.
DDS_DataReaderResourceLimitsQosPolicy (p. 867)	**lazily_allocate_batch_resources** (p. 885)
	*<<**extension**>>* *(p. 213)* Determines whether the **DDSDataReader** (p. 1556) pre-allocates resources for receiving batches of samples or lazily allocates them when needed.
DDS_DataWriterResourceLimitsQosPolicy (p. 917)	**initial_concurrent_blocking_threads** (p. 919)
	[DEPRECATED] (p. **384**) *<<**extension**>>* *(p. 213)* The initial number of threads that are allowed to concurrently block on write call on the same **DDSDataWriter** (p. 1589).
DDS_DataWriterResourceLimitsQosPolicy (p. 917)	**max_concurrent_blocking_threads** (p. 919)
	[DEPRECATED] (p. **384**) *<<**extension**>>* *(p. 213)* The maximum number of threads that are allowed to concurrently block on write call on the same **DDSDataWriter** (p. 1589).
DDS_DataWriterResourceLimitsQosPolicy (p. 917)	**max_remote_reader_filters** (p. 920)
	*<<**extension**>>* *(p. 213)* The maximum number of remote DataReaders for which the **DDSDataWriter** (p. 1589) will perform content-based filtering.
DDS_DataWriterResourceLimitsQosPolicy (p. 917)	**initial_batches** (p. 920)
	*<<**extension**>>* *(p. 213)* Represents the initial number of batches a **DDSDataWriter** (p. 1589) will manage.
DDS_DataWriterResourceLimitsQosPolicy (p. 917)	**max_batches** (p. 921)
	*<<**extension**>>* *(p. 213)* Represents the maximum number of batches a **DDSDataWriter** (p. 1589) will manage.

DDS_DataWriterResourceLimitsQosPolicy (p. 917)	**instance_replacement** (p. 921)
	<<***extension***>> *(p. 213)* Sets the kinds of instances allowed to be replaced when instance resource limits are reached.
DDS_DataWriterResourceLimitsQosPolicy (p. 917)	**replace_empty_instances** (p. 922)
	<<***extension***>> *(p. 213)* Whether or not to replace empty instances during instance replacement
DDS_DataWriterResourceLimitsQosPolicy (p. 917)	**autoregister_instances** (p. 922)
	[DEPRECATED] (p. 384) <<***extension***>> *(p. 213)* Whether or not to automatically register new instances
DDS_DataWriterResourceLimitsQosPolicy (p. 917)	**max_virtual_writers** (p. 923)
	<<***extension***>> *(p. 213)* The maximum number of virtual writers supported by a **DDSDataWriter** (p. 1589)
DDS_DataWriterResourceLimitsQosPolicy (p. 917)	**initial_virtual_writers** (p. 923)
	<<***extension***>> *(p. 213)* The initial number of virtual writers supported by a **DDSDataWriter** (p. 1589)
DDS_DataWriterResourceLimitsQosPolicy (p. 917)	**initial_remote_readers** (p. 926)
	<<***extension***>> *(p. 213)* Controls the initial memory allocation for the matched remote readers of a **DDSDataWriter** (p. 1589).
DDS_DataWriterResourceLimitsQosPolicy (p. 917)	**max_remote_readers** (p. 923)
	<<***extension***>> *(p. 213)* The maximum number of remote readers supported by a **DDSDataWriter** (p. 1589)
DDS_DataWriterResourceLimitsQosPolicy (p. 917)	**max_app_ack_remote_readers** (p. 924)
	<<***extension***>> *(p. 213)* The maximum number of application-level acknowledging remote readers supported by a **DDSDataWriter** (p. 1589)
DDS_DataWriterResourceLimitsQosPolicy (p. 917)	**initial_active_topic_queries** (p. 924)
	<<***extension***>> *(p. 213)* Represents the initial number of active topic queries a **DDSDataWriter** (p. 1589) will manage.
DDS_DataWriterResourceLimitsQosPolicy (p. 917)	**max_active_topic_queries** (p. 924)
	<<***extension***>> *(p. 213)* Represents the maximum number of active topic queries a **DDSDataWriter** (p. 1589) will manage.
DDS_DataWriterResourceLimitsQosPolicy (p. 917)	**writer_loaned_sample_allocation** (p. 925)
	<<***extension***>> *(p. 213)* Represents the allocation settings of loaned samples managed by a **DDSData↩ Writer** (p. 1589).

DDS_DataWriterResourceLimitsQosPolicy (p. 917)	**initialize_writer_loaned_sample** (p. 925)
	<<***extension***>> *(p. 213)* Whether or not to initialize loaned samples returned by a **DDSDataWriter** (p. 1589)
DDS_DataWriterTransferModeQosPolicy (p. 927)	**shmem_ref_settings** (p. 928)
	<<***extension***>> *(p. 213)* Settings related to transferring data using shared memory references.
DDS_DataReaderProtocolQosPolicy (p. 843)	**virtual_guid** (p. 844)
	<<***extension***>> *(p. 213)* The virtual GUID (Global Unique Identifier).
DDS_DataReaderProtocolQosPolicy (p. 843)	**rtps_object_id** (p. 844)
	<<***extension***>> *(p. 213)* The RTPS Object ID.
DDS_DataReaderProtocolQosPolicy (p. 843)	**rtps_reliable_reader** (p. 846)
	<<***extension***>> *(p. 213)* RTPS protocol-related configuration settings for the RTPS reliable reader associated with a **DDSDataReader** (p. 1556). This parameter only has effect if the reader is configured with **DDS_RELIABLE_RELIABILITY_QOS** (p. 633) **DDS_↩ ReliabilityQosPolicyKind** (p. 632).
DDS_DataReaderProtocolQosPolicy (p. 843)	**expects_inline_qos** (p. 844)
	[DEPRECATED] (p. 384) <<***extension***>> *(p. 213)* Specifies whether this DataReader expects inline QoS with every sample.
DDS_DataReaderProtocolQosPolicy (p. 843)	**disable_positive_acks** (p. 845)
	<<***extension***>> *(p. 213)* Whether the reader sends positive acknowledgements to writers.
DDS_DataReaderProtocolQosPolicy (p. 843)	**propagate_dispose_of_unregistered_instances** (p. 845)
	<<***extension***>> *(p. 213)* Indicates whether or not an instance can move to the **DDS_NOT_ALIVE_↩ DISPOSED_INSTANCE_STATE** (p. 166) state without being in the **DDS_ALIVE_INSTANCE_STATE** (p. 166) state.
DDS_DataReaderProtocolQosPolicy (p. 843)	**propagate_unregister_of_disposed_instances** (p. 846)
	<<***extension***>> *(p. 213)* Indicates whether or not an instance can move to the **DDS_NOT_ALIVE_NO↩ _WRITERS_INSTANCE_STATE** (p. 166) state directly from the **DDS_NOT_ALIVE_DISPOSED_INSTANCE↩ _STATE** (p. 166).
DDS_DataWriterProtocolQosPolicy (p. 892)	**virtual_guid** (p. 893)
	<<***extension***>> *(p. 213)* The virtual GUID (Global Unique Identifier).

DDS_DataWriterProtocolQosPolicy (p. 892)	**rtps_object_id** (p. 893)
	<<***extension***>> *(p. 213)* The RTPS Object ID.
DDS_DataWriterProtocolQosPolicy (p. 892)	**push_on_write** (p. 894)
	<<***extension***>> *(p. 213)* Whether to push sample out when write is called.
DDS_DataWriterProtocolQosPolicy (p. 892)	**disable_positive_acks** (p. 894)
	<<***extension***>> *(p. 213)* Controls whether or not the writer expects positive acknowledgements from matching readers.
DDS_DataWriterProtocolQosPolicy (p. 892)	**disable_inline_keyhash** (p. 895)
	<<***extension***>> *(p. 213)* Controls whether or not a keyhash is propagated on the wire with each sample.
DDS_DataWriterProtocolQosPolicy (p. 892)	**serialize_key_with_dispose** (p. 895)
	<<***extension***>> *(p. 213)* Controls whether or not the serialized key is propagated on the wire with dispose samples.
DDS_DataWriterProtocolQosPolicy (p. 892)	**propagate_app_ack_with_no_response** (p. 896)
	<<***extension***>> *(p. 213)* Controls whether or not a **DDSDataWriter** (p. 1589) receives **DDSDataWriter**↩ **Listener::on_application_acknowledgment** (p. 1617) notifications with an empty or invalid response.
DDS_DataWriterProtocolQosPolicy (p. 892)	**rtps_reliable_writer** (p. 896)
	<<***extension***>> *(p. 213)* RTPS protocol-related configuration settings for the RTPS reliable writer associated with a **DDSDataWriter** (p. 1589). This parameter only has effect if both the writer and the matching reader are configured with **DDS_RELIABLE_RELIABILITY_**↩ **QOS** (p. 633) **DDS_ReliabilityQosPolicyKind** (p. 632).
DDS_DataWriterProtocolQosPolicy (p. 892)	**initial_virtual_sequence_number** (p. 897)
	<<***extension***>> *(p. 213)* Determines, the initial virtual sequence number for this DataWriter.
DDS_SystemResourceLimitsQosPolicy (p. 1382)	**max_objects_per_thread** (p. 1383)
	[DEPRECATED] (p. **384)** <<***extension***>> *(p. 213)* The maximum number of objects that can be stored per thread for a **DDSDomainParticipantFactory** (p. 1694).
DDS_SystemResourceLimitsQosPolicy (p. 1382)	**initial_objects_per_thread** (p. 1383)
	<<***extension***>> *(p. 213)* The number of objects per thread for a **DDSDomainParticipantFactory** (p. 1694) for which infrastructure will initially be allocated.

DDS_DomainParticipantResourceLimitsQosPolicy (p. 977)	**local_writer_allocation** (p. 981)
	<<***extension***>> *(p. 213)* Allocation settings applied to local DataWriters.
DDS_DomainParticipantResourceLimitsQosPolicy (p. 977)	**local_reader_allocation** (p. 981)
	<<***extension***>> *(p. 213)* Allocation settings applied to local DataReaders.
DDS_DomainParticipantResourceLimitsQosPolicy (p. 977)	**local_publisher_allocation** (p. 981)
	<<***extension***>> *(p. 213)* Allocation settings applied to local Publisher.
DDS_DomainParticipantResourceLimitsQosPolicy (p. 977)	**local_subscriber_allocation** (p. 981)
	<<***extension***>> *(p. 213)* Allocation settings applied to local Subscriber.
DDS_DomainParticipantResourceLimitsQosPolicy (p. 977)	**local_topic_allocation** (p. 982)
	<<***extension***>> *(p. 213)* Allocation settings applied to local Topic.
DDS_DomainParticipantResourceLimitsQosPolicy (p. 977)	**remote_writer_allocation** (p. 982)
	<<***extension***>> *(p. 213)* Allocation settings applied to remote DataWriters.
DDS_DomainParticipantResourceLimitsQosPolicy (p. 977)	**remote_reader_allocation** (p. 982)
	<<***extension***>> *(p. 213)* Allocation settings applied to remote DataReaders.
DDS_DomainParticipantResourceLimitsQosPolicy (p. 977)	**remote_participant_allocation** (p. 983)
	<<***extension***>> *(p. 213)* Allocation settings applied to remote DomainParticipants.
DDS_DomainParticipantResourceLimitsQosPolicy (p. 977)	**matching_writer_reader_pair_allocation** (p. 983)
	<<***extension***>> *(p. 213)* Allocation settings applied to matching local writer and remote/local reader pairs.
DDS_DomainParticipantResourceLimitsQosPolicy (p. 977)	**matching_reader_writer_pair_allocation** (p. 983)
	<<***extension***>> *(p. 213)* Allocation settings applied to matching local reader and remote/local writer pairs.

DDS_DomainParticipantResourceLimitsQosPolicy (p. 977)	**ignored_entity_allocation** (p. 984)
	<<***extension***>> *(p. 213)* Allocation settings applied to ignored entities.
DDS_DomainParticipantResourceLimitsQosPolicy (p. 977)	**content_filtered_topic_allocation** (p. 984)
	<<***extension***>> *(p. 213)* Allocation settings applied to content filtered topic.
DDS_DomainParticipantResourceLimitsQosPolicy (p. 977)	**content_filter_allocation** (p. 984)
	<<***extension***>> *(p. 213)* Allocation settings applied to content filter.
DDS_DomainParticipantResourceLimitsQosPolicy (p. 977)	**read_condition_allocation** (p. 985)
	<<***extension***>> *(p. 213)* Allocation settings applied to read condition pool.
DDS_DomainParticipantResourceLimitsQosPolicy (p. 977)	**query_condition_allocation** (p. 985)
	<<***extension***>> *(p. 213)* Allocation settings applied to query condition pool.
DDS_DomainParticipantResourceLimitsQosPolicy (p. 977)	**outstanding_asynchronous_sample_allocation** (p. 985)
	<<***extension***>> *(p. 213)* Allocation settings applied to the maximum number of samples (from all **DDSData↩ Writer** (p. 1589)) waiting to be asynchronously written.
DDS_DomainParticipantResourceLimitsQosPolicy (p. 977)	**flow_controller_allocation** (p. 986)
	<<***extension***>> *(p. 213)* Allocation settings applied to flow controllers.
DDS_DomainParticipantResourceLimitsQosPolicy (p. 977)	**local_writer_hash_buckets** (p. 986)
	[DEPRECATED] (p. 384) <<***extension***>> *(p. 213)* Hash_Buckets settings applied to local DataWriters.
DDS_DomainParticipantResourceLimitsQosPolicy (p. 977)	**local_reader_hash_buckets** (p. 986)
	[DEPRECATED] (p. 384) <<***extension***>> *(p. 213)* Number of hash buckets for local DataReaders.
DDS_DomainParticipantResourceLimitsQosPolicy (p. 977)	**local_publisher_hash_buckets** (p. 987)
	[DEPRECATED] (p. 384) <<***extension***>> *(p. 213)* Number of hash buckets for local Publisher.

DDS_DomainParticipantResourceLimitsQosPolicy (p. 977)	**local_subscriber_hash_buckets** (p. 987)
	[DEPRECATED] (p. **384**) <<***extension***>> (p. *213*) Number of hash buckets for local Subscriber.
DDS_DomainParticipantResourceLimitsQosPolicy (p. 977)	**local_topic_hash_buckets** (p. 987)
	[DEPRECATED] (p. **384**) <<***extension***>> (p. *213*) Number of hash buckets for local Topic.
DDS_DomainParticipantResourceLimitsQosPolicy (p. 977)	**remote_writer_hash_buckets** (p. 988)
	[DEPRECATED] (p. **384**) <<***extension***>> (p. *213*) Number of hash buckets for remote DataWriters.
DDS_DomainParticipantResourceLimitsQosPolicy (p. 977)	**remote_reader_hash_buckets** (p. 988)
	[DEPRECATED] (p. **384**) <<***extension***>> (p. *213*) Number of hash buckets for remote DataReaders.
DDS_DomainParticipantResourceLimitsQosPolicy (p. 977)	**remote_participant_hash_buckets** (p. 988)
	[DEPRECATED] (p. **384**) <<***extension***>> (p. *213*) Number of hash buckets for remote DomainParticipants.
DDS_DomainParticipantResourceLimitsQosPolicy (p. 977)	**matching_writer_reader_pair_hash_buckets** (p. 989)
	[DEPRECATED] (p. **384**) <<***extension***>> (p. *213*) Number of hash buckets for matching local writer and remote/local reader pairs.
DDS_DomainParticipantResourceLimitsQosPolicy (p. 977)	**matching_reader_writer_pair_hash_buckets** (p. 989)
	[DEPRECATED] (p. **384**) <<***extension***>> (p. *213*) Number of hash buckets for matching local reader and remote/local writer pairs.
DDS_DomainParticipantResourceLimitsQosPolicy (p. 977)	**ignored_entity_hash_buckets** (p. 989)
	[DEPRECATED] (p. **384**) <<***extension***>> (p. *213*) Number of hash buckets for ignored entities.
DDS_DomainParticipantResourceLimitsQosPolicy (p. 977)	**content_filtered_topic_hash_buckets** (p. 990)
	[DEPRECATED] (p. **384**) <<***extension***>> (p. *213*) Number of hash buckets for content filtered topics.
DDS_DomainParticipantResourceLimitsQosPolicy (p. 977)	**content_filter_hash_buckets** (p. 990)
	[DEPRECATED] (p. **384**) <<***extension***>> (p. *213*) Number of hash buckets for content filters.

DDS_DomainParticipantResourceLimitsQosPolicy (p. 977)	**flow_controller_hash_buckets** (p. 990)
	[DEPRECATED] (p. 384) <<***extension***>> *(p. 213)* Number of hash buckets for flow controllers.
DDS_DomainParticipantResourceLimitsQosPolicy (p. 977)	**max_gather_destinations** (p. 991)
	<<***extension***>> *(p. 213)* Maximum number of destinations per RTI Connext send.
DDS_DomainParticipantResourceLimitsQosPolicy (p. 977)	**participant_user_data_max_length** (p. 991)
	<<***extension***>> *(p. 213)* Maximum length of user data in **DDS_DomainParticipantQos** (p. 971) and **DDS_↩ ParticipantBuiltinTopicData** (p. 1229)
DDS_DomainParticipantResourceLimitsQosPolicy (p. 977)	**topic_data_max_length** (p. 991)
	<<***extension***>> *(p. 213)* Maximum length of topic data in **DDS_TopicQos** (p. 1397), **DDS_TopicBuiltin↩ TopicData** (p. 1391), **DDS_PublicationBuiltinTopic↩ Data** (p. 1264) and **DDS_SubscriptionBuiltinTopic↩ Data** (p. 1370)
DDS_DomainParticipantResourceLimitsQosPolicy (p. 977)	**publisher_group_data_max_length** (p. 992)
	<<***extension***>> *(p. 213)* Maximum length of group data in **DDS_PublisherQos** (p. 1276) and **DDS_↩ PublicationBuiltinTopicData** (p. 1264)
DDS_DomainParticipantResourceLimitsQosPolicy (p. 977)	**subscriber_group_data_max_length** (p. 992)
	<<***extension***>> *(p. 213)* Maximum length of group data in **DDS_SubscriberQos** (p. 1367) and **DDS_↩ SubscriptionBuiltinTopicData** (p. 1370)
DDS_DomainParticipantResourceLimitsQosPolicy (p. 977)	**writer_user_data_max_length** (p. 992)
	<<***extension***>> *(p. 213)* Maximum length of user data in **DDS_DataWriterQos** (p. 909) and **DDS_↩ PublicationBuiltinTopicData** (p. 1264)
DDS_DomainParticipantResourceLimitsQosPolicy (p. 977)	**reader_user_data_max_length** (p. 993)
	<<***extension***>> *(p. 213)* Maximum length of user data in **DDS_DataReaderQos** (p. 858) and **DDS_↩ SubscriptionBuiltinTopicData** (p. 1370)
DDS_DomainParticipantResourceLimitsQosPolicy (p. 977)	**max_partitions** (p. 993)

	<<***extension***>> *(p. 213)* Maximum number of partition name strings allowable in a **DDS_PartitionQosPolicy** (p. 1240).
DDS_DomainParticipantResourceLimitsQosPolicy (p. 977)	**max_partition_cumulative_characters** (p. 993)
	<<***extension***>> *(p. 213)* Maximum number of combined characters allowable in all partition names in a **DDS_PartitionQosPolicy** (p. 1240).
DDS_DomainParticipantResourceLimitsQosPolicy (p. 977)	**type_code_max_serialized_length** (p. 994)
	[DEPRECATED] (p. 384) <<***extension***>> *(p. 213)* Maximum size of serialized string for type code.
DDS_DomainParticipantResourceLimitsQosPolicy (p. 977)	**type_object_max_serialized_length** (p. 994)
	<<***extension***>> *(p. 213)* The maximum length, in bytes, that the buffer to serialize a TypeObject v1 can consume.
DDS_DomainParticipantResourceLimitsQosPolicy (p. 977)	**type_object_max_deserialized_length** (p. 996)
	<<***extension***>> *(p. 213)* The maximum number of bytes that a deserialized TypeObject v1 can consume.
DDS_DomainParticipantResourceLimitsQosPolicy (p. 977)	**deserialized_type_object_dynamic_allocation_threshold** (p. 996)
	<<***extension***>> *(p. 213)* A threshold, in bytes, for dynamic memory allocation for the deserialized TypeObject v1.
DDS_DomainParticipantResourceLimitsQosPolicy (p. 977)	**serialized_type_object_dynamic_allocation_threshold** (p. 995)
	<<***extension***>> *(p. 213)* A threshold, in bytes, for dynamic memory allocation for the serialized TypeObject v1.
DDS_DomainParticipantResourceLimitsQosPolicy (p. 977)	**contentfilter_property_max_length** (p. 997)
	<<***extension***>> *(p. 213)* This field is the maximum length of all data related to a Content-filtered topic.
DDS_DomainParticipantResourceLimitsQosPolicy (p. 977)	**channel_seq_max_length** (p. 997)
	<<***extension***>> *(p. 213)* Maximum number of channels that can be specified in **DDS_MultiChannelQosPolicy** (p. 1214) for MultiChannel DataWriters.
DDS_DomainParticipantResourceLimitsQosPolicy (p. 977)	**channel_filter_expression_max_length** (p. 998)

	<<***extension***>> *(p. 213)* Maximum length of a channel **DDS_ChannelSettings_t::filter_expression** (p. 822) in a MultiChannel DataWriter.
DDS_DomainParticipantResourceLimitsQosPolicy (p. 977)	**participant_property_list_max_length** (p. 998)
	<<***extension***>> *(p. 213)* Maximum number of properties associated with the **DDSDomainParticipant** (p. 1619).
DDS_DomainParticipantResourceLimitsQosPolicy (p. 977)	**participant_property_string_max_length** (p. 998)
	<<***extension***>> *(p. 213)* Maximum string length of the properties associated with the **DDSDomainParticipant** (p. 1619).
DDS_DomainParticipantResourceLimitsQosPolicy (p. 977)	**writer_property_list_max_length** (p. 999)
	<<***extension***>> *(p. 213)* Maximum number of properties associated with a **DDSDataWriter** (p. 1589).
DDS_DomainParticipantResourceLimitsQosPolicy (p. 977)	**writer_property_string_max_length** (p. 999)
	<<***extension***>> *(p. 213)* Maximum string length of the properties associated with a **DDSDataWriter** (p. 1589).
DDS_DomainParticipantResourceLimitsQosPolicy (p. 977)	**reader_property_list_max_length** (p. 999)
	<<***extension***>> *(p. 213)* Maximum number of properties associated with a **DDSDataReader** (p. 1556).
DDS_DomainParticipantResourceLimitsQosPolicy (p. 977)	**reader_property_string_max_length** (p. 1000)
	<<***extension***>> *(p. 213)* Maximum string length of the properties associated with a **DDSDataReader** (p. 1556).
DDS_DomainParticipantResourceLimitsQosPolicy (p. 977)	**max_endpoint_groups** (p. 1000)
	<<***extension***>> *(p. 213)* Maximum number of **DDS_EndpointGroup_t** (p. 1142) allowable in a **DDS_AvailabilityQosPolicy** (p. 805).
DDS_DomainParticipantResourceLimitsQosPolicy (p. 977)	**max_endpoint_group_cumulative_characters** (p. 1000)
	<<***extension***>> *(p. 213)* Maximum number of combined role_name characters allowed in all **DDS_EndpointGroup_t** (p. 1142) in a **DDS_AvailabilityQosPolicy** (p. 805).
DDS_DomainParticipantResourceLimitsQosPolicy (p. 977)	**transport_info_list_max_length** (p. 1001)

	<<***extension***>> *(p. 213)* Maximum number of installed transports to send and receive information about in **DDS_ParticipantBuiltinTopicData::transport_info** (p. 1232).
DDS_DomainParticipantResourceLimitsQosPolicy (p. 977)	**ignored_entity_replacement_kind** (p. 1001)
	[DEPRECATED] (p. **384**) <<***extension***>> *(p. 213)* Replacement policy for the ignored entities. It sets what entity can be replaced when resource limits set in **DDS_DomainParticipantResourceLimitsQosPolicy**↩ **::ignored_entity_allocation** (p. 984) are reached.
DDS_DomainParticipantResourceLimitsQosPolicy (p. 977)	**remote_topic_query_allocation** (p. 1001)
	<<***extension***>> *(p. 213)* Allocation settings applied to remote TopicQueries.
DDS_DomainParticipantResourceLimitsQosPolicy (p. 977)	**remote_topic_query_hash_buckets** (p. 1002)
	[DEPRECATED] (p. **384**) <<***extension***>> *(p. 213)* Number of hash buckets for remote TopicQueries.
DDS_DomainParticipantResourceLimitsQosPolicy (p. 977)	**writer_data_tag_list_max_length** (p. 1002)
	<<***extension***>> *(p. 213)* Maximum number of data tags associated with a **DDSDataWriter** (p. 1589).
DDS_DomainParticipantResourceLimitsQosPolicy (p. 977)	**writer_data_tag_string_max_length** (p. 1003)
	<<***extension***>> *(p. 213)* Maximum string length of the data tags associated with a **DDSDataWriter** (p. 1589).
DDS_DomainParticipantResourceLimitsQosPolicy (p. 977)	**reader_data_tag_list_max_length** (p. 1003)
	<<***extension***>> *(p. 213)* Maximum number of data tags associated with a **DDSDataReader** (p. 1556).
DDS_DomainParticipantResourceLimitsQosPolicy (p. 977)	**reader_data_tag_string_max_length** (p. 1003)
	<<***extension***>> *(p. 213)* Maximum string length of the data tags associated with a **DDSDataReader** (p. 1556).
DDS_DomainParticipantResourceLimitsQosPolicy (p. 977)	**shmem_ref_transfer_mode_max_segments** (p. 1004)
	<<***extension***>> *(p. 213)* Maximum number of segments created by all DataWriters belonging to a **DDSDomainParticipant** (p. 1619).
DDS_EventQosPolicy (p. 1150)	**thread** (p. 1151)
	<<***extension***>> *(p. 213)* Event thread QoS.

DDS_EventQosPolicy (p. 1150)	**initial_count** (p. 1151)
	<<***extension***>> *(p. 213)* The initial number of events.
DDS_EventQosPolicy (p. 1150)	**max_count** (p. 1152)
	<<***extension***>> *(p. 213)* The maximum number of events.
DDS_DatabaseQosPolicy (p. 831)	**thread** (p. 832)
	<<***extension***>> *(p. 213)* Database thread settings.
DDS_DatabaseQosPolicy (p. 831)	**shutdown_timeout** (p. 832)
	[DEPRECATED] (p. **384)** <<***extension***>> *(p. 213)* The maximum wait time during a shutdown.
DDS_DatabaseQosPolicy (p. 831)	**cleanup_period** (p. 833)
	[DEPRECATED] (p. **384)** <<***extension***>> *(p. 213)* The period at which the database thread wakes up to remove deleted records.
DDS_DatabaseQosPolicy (p. 831)	**shutdown_cleanup_period** (p. 833)
	[DEPRECATED] (p. **384)** <<***extension***>> *(p. 213)* The clean-up period used during database shut-down.
DDS_DatabaseQosPolicy (p. 831)	**initial_records** (p. 833)
	<<***extension***>> *(p. 213)* The initial number of total records.
DDS_DatabaseQosPolicy (p. 831)	**max_skiplist_level** (p. 834)
	[DEPRECATED] (p. **384)** <<***extension***>> *(p. 213)* The maximum level of the skiplist.
DDS_DatabaseQosPolicy (p. 831)	**initial_weak_references** (p. 835)
	<<***extension***>> *(p. 213)* The initial number of weak references.
DDS_DatabaseQosPolicy (p. 831)	**max_weak_references** (p. 834)
	<<***extension***>> *(p. 213)* The maximum number of weak references.
DDS_ReceiverPoolQosPolicy (p. 1293)	**thread** (p. 1294)
	<<***extension***>> *(p. 213)* Receiver pool thread(s).
DDS_ReceiverPoolQosPolicy (p. 1293)	**buffer_size** (p. 1294)
	<<***extension***>> *(p. 213)* The receive buffer size in bytes.
DDS_ReceiverPoolQosPolicy (p. 1293)	**buffer_alignment** (p. 1295)

	[DEPRECATED] (p. **384**) <<***extension***>> *(p. 213)* The receive buffer alignment.
DDS_DiscoveryConfigQosPolicy (p. 934)	**remote_participant_purge_kind** (p. 938)
	[DEPRECATED] (p. **384**) <<***extension***>> *(p. 213)* The participant's behavior for maintaining knowledge of remote participants (and their contained entities) with which discovery communication has been lost.
DDS_DiscoveryConfigQosPolicy (p. 934)	**max_liveliness_loss_detection_period** (p. 938)
	[DEPRECATED] (p. **384**) <<***extension***>> *(p. 213)* The maximum amount of time between when a remote entity stops maintaining its liveliness and when the matched local entity realizes that fact.
DDS_DiscoveryConfigQosPolicy (p. 934)	**participant_reader_resource_limits** (p. 941)
	<<***extension***>> *(p. 213)* Resource limits.
DDS_DiscoveryConfigQosPolicy (p. 934)	**publication_reader** (p. 941)
	<<***extension***>> *(p. 213)* RTPS protocol-related configuration settings for the RTPS reliable reader associated with a built-in publication reader.
DDS_DiscoveryConfigQosPolicy (p. 934)	**publication_reader_resource_limits** (p. 942)
	<<***extension***>> *(p. 213)* Resource limits.
DDS_DiscoveryConfigQosPolicy (p. 934)	**subscription_reader** (p. 942)
	<<***extension***>> *(p. 213)* RTPS protocol-related configuration settings for the RTPS reliable reader associated with a built-in subscription reader.
DDS_DiscoveryConfigQosPolicy (p. 934)	**subscription_reader_resource_limits** (p. 943)
	<<***extension***>> *(p. 213)* Resource limits.
DDS_DiscoveryConfigQosPolicy (p. 934)	**participant_message_writer** (p. 947)
	<<***extension***>> *(p. 213)* RTPS protocol-related configuration settings for the RTPS reliable writer associated with the built-in participant message writer. This parameter only has effect if the matching participant message reader is configured with **DDS_RELIABLE↩ _RELIABILITY_QOS** (p. 633) **DDS_ReliabilityQos↩ PolicyKind** (p. 632).
DDS_DiscoveryConfigQosPolicy (p. 934)	**publication_writer_publish_mode** (p. 948)
	<<***extension***>> *(p. 213)* Publish mode policy for the built-in publication writer.
DDS_DiscoveryConfigQosPolicy (p. 934)	**subscription_writer_publish_mode** (p. 949)
	<<***extension***>> *(p. 213)* Publish mode policy for the built-in subscription writer.

DDS_DiscoveryConfigQosPolicy (p. 934)	**asynchronous_publisher** (p. 949)
	<<***extension***>> *(p. 213)* Asynchronous publishing settings for the discovery **DDSPublisher** (p. 1818) and all entities that are created by it.
DDS_DiscoveryConfigQosPolicy (p. 934)	**default_domain_announcement_period** (p. 949)
	<<***extension***>> *(p. 213)* The period to announce a participant to the default domain 0.
DDS_DiscoveryConfigQosPolicy (p. 934)	**ignore_default_domain_announcements** (p. 950)
	[DEPRECATED] (p. 384) <<***extension***>> *(p. 213)* Used to ignore the announcements received by a participant on the default domain 0 corresponding to participants running on domains IDs other than 0.
DDS_DiscoveryConfigQosPolicy (p. 934)	**participant_message_reader_reliability_kind** (p. 946)
	[DEPRECATED] (p. 384) <<***extension***>> *(p. 213)* Reliability policy for a built-in participant message reader.
DDS_DiscoveryConfigQosPolicy (p. 934)	**participant_message_reader** (p. 947)
	<<***extension***>> *(p. 213)* RTPS reliable reader protocol-related configuration settings for a built-in participant message reader. This parameter only has effect if **DDS_DiscoveryConfigQosPolicy::participant_message_reader_reliability_kind** (p. 946) is set to **DDS_RELIABLE_RELIABILITY_QOS** (p. 633).
DDS_DiscoveryConfigQosPolicy (p. 934)	**publication_writer** (p. 943)
	<<***extension***>> *(p. 213)* RTPS protocol-related configuration settings for the RTPS reliable writer associated with a built-in publication writer.
DDS_DiscoveryConfigQosPolicy (p. 934)	**publication_writer_data_lifecycle** (p. 944)
	<<***extension***>> *(p. 213)* Writer data lifecycle settings for a built-in publication writer.
DDS_DiscoveryConfigQosPolicy (p. 934)	**subscription_writer** (p. 945)
	<<***extension***>> *(p. 213)* RTPS protocol-related configuration settings for the RTPS reliable writer associated with a built-in subscription writer.
DDS_DiscoveryConfigQosPolicy (p. 934)	**subscription_writer_data_lifecycle** (p. 945)
	<<***extension***>> *(p. 213)* Writer data lifecycle settings for a built-in subscription writer.
DDS_DiscoveryConfigQosPolicy (p. 934)	**builtin_discovery_plugins** (p. 946)
	<<***extension***>> *(p. 213)* Mask of built-in discovery plugin kinds.

DDS_DiscoveryConfigQosPolicy (p. 934)	**enabled_builtin_channels** (p. 946)
	<<***extension***>> *(p. 213)* The mask specifying which built-in channels should be enabled.
DDS_DiscoveryConfigQosPolicy (p. 934)	**service_request_writer** (p. 950)
	<<***extension***>> *(p. 213)* RTPS protocol-related configuration settings for the RTPS reliable writer associated with the built-in **DDS_ServiceRequest** (p. 1359) writer.
DDS_DiscoveryConfigQosPolicy (p. 934)	**service_request_writer_data_lifecycle** (p. 951)
	<<***extension***>> *(p. 213)* Writer data lifecycle settings for a built-in **DDS_ServiceRequest** (p. 1359) writer.
DDS_DiscoveryConfigQosPolicy (p. 934)	**service_request_writer_publish_mode** (p. 951)
	<<***extension***>> *(p. 213)* Publish mode policy for the built-in service request writer.
DDS_DiscoveryConfigQosPolicy (p. 934)	**service_request_reader** (p. 952)
	<<***extension***>> *(p. 213)* RTPS reliable reader protocol-related configuration settings for a built-in **DDS_ServiceRequest** (p. 1359) reader.
DDS_DiscoveryConfigQosPolicy (p. 934)	**locator_reachability_assert_period** (p. 952)
	<<***extension***>> *(p. 213)* Period at which this DomainParticipant will assert the locators discovered from other DomainParticipants.
DDS_DiscoveryConfigQosPolicy (p. 934)	**locator_reachability_change_detection_period** (p. 953)
	<<***extension***>> *(p. 213)* Period at which this DomainParticipant will check if its locators are reachable from other DomainParticipants.
DDS_DiscoveryConfigQosPolicy (p. 934)	**secure_volatile_writer** (p. 954)
	<<***extension***>> *(p. 213)* RTPS protocol-related configuration settings for the RTPS reliable writer associated with the built-in secure volatile writer.
DDS_DiscoveryConfigQosPolicy (p. 934)	**secure_volatile_writer_publish_mode** (p. 955)
	<<***extension***>> *(p. 213)* Publish mode policy for the built-in secure volatile writer.
DDS_DiscoveryConfigQosPolicy (p. 934)	**secure_volatile_reader** (p. 955)
	<<***extension***>> *(p. 213)* RTPS reliable reader protocol-related configuration settings for the built-in secure volatile reader.
DDS_DiscoveryConfigQosPolicy (p. 934)	**endpoint_type_object_lb_serialization_threshold** (p. 956)

	<<***extension***>> *(p. 213)* Controls when/if the Type↩ Object v1 is compressed when propagated in Simple Endpoint Discovery.
DDS_DiscoveryConfigQosPolicy (p. 934)	**dns_tracker_polling_period** (p. 956)
	<<***extension***>> *(p. 213)* Duration that specifies the period used by the DNS tracker to poll the DNS service and check for changes in the hostnames.
DDS_DiscoveryConfigQosPolicy (p. 934)	**participant_configuration_reader** (p. 959)
	<<***extension***>> *(p. 213)* RTPS protocol-related configuration settings for the RTPS reliable reader associated with a built-in participant configuration reader.
DDS_DiscoveryConfigQosPolicy (p. 934)	**participant_configuration_reader_resource_limits** (p. 959)
	<<***extension***>> *(p. 213)* Resource limits for the built-in topic participant configuration reader.
DDS_DiscoveryConfigQosPolicy (p. 934)	**participant_configuration_writer_publish_mode** (p. 957)
	<<***extension***>> *(p. 213)* Publish mode policy for the built-in participant configuration writer.
DDS_DiscoveryConfigQosPolicy (p. 934)	**participant_configuration_writer** (p. 957)
	<<***extension***>> *(p. 213)* RTPS protocol-related configuration settings for the RTPS reliable writer associated with a built-in participant configuration writer.
DDS_DiscoveryConfigQosPolicy (p. 934)	**participant_configuration_writer_data_lifecycle** (p. 958)
	<<***extension***>> *(p. 213)* Writer data lifecycle settings for a built-in participant configuration writer.
DDS_DiscoveryConfigQosPolicy (p. 934)	**request_types_filter** (p. 959)
	<<***extension***>> *(p. 213)* Topic name pattern for requesting all types via Type Lookup Service.
DDS_RtpsReliableReaderProtocol_t (p. 1314)	**min_heartbeat_response_delay** (p. 1315)
	<<***extension***>> *(p. 213)* The minimum delay to respond to a heartbeat.
DDS_RtpsReliableReaderProtocol_t (p. 1314)	**max_heartbeat_response_delay** (p. 1315)
	<<***extension***>> *(p. 213)* The maximum delay to respond to a heartbeat.
DDS_RtpsReliableReaderProtocol_t (p. 1314)	**heartbeat_suppression_duration** (p. 1316)
	<<***extension***>> *(p. 213)* The duration a reader ignores consecutively received heartbeats

DDS_RtpsReliableReaderProtocol_t (p. 1314)	**nack_period** (p. 1316)
	<<***extension***>> *(p. 213)* The period at which to send NACKs
DDS_RtpsReliableReaderProtocol_t (p. 1314)	**receive_window_size** (p. 1317)
	<<***extension***>> *(p. 213)* The number of received out-of-order samples a reader can keep at a time
DDS_RtpsReliableReaderProtocol_t (p. 1314)	**round_trip_time** (p. 1317)
	<<***extension***>> *(p. 213)* The duration from sending a NACK to receiving a repair of a sample
DDS_RtpsReliableReaderProtocol_t (p. 1314)	**app_ack_period** (p. 1318)
	<<***extension***>> *(p. 213)* The period at which application-level acknowledgment messages are sent
DDS_RtpsReliableReaderProtocol_t (p. 1314)	**min_app_ack_response_keep_duration** (p. 1318)
	<<***extension***>> *(p. 213)* Minimum duration for which application-level acknowledgment response data is kept
DDS_RtpsReliableReaderProtocol_t (p. 1314)	**samples_per_app_ack** (p. 1319)
	<<***extension***>> *(p. 213)* The minimum number of samples acknowledged by one application-level acknowledgment message
DDS_RtpsReliableWriterProtocol_t (p. 1320)	**low_watermark** (p. 1321)
	<<***extension***>> *(p. 213)* When the number of unacknowledged samples in the current send window of a reliable writer meets or falls below this threshold, the **DDS_RELIABLE_WRITER_CACHE_CHANGED↩ _STATUS** (p. 545) is considered to have changed.
DDS_RtpsReliableWriterProtocol_t (p. 1320)	**high_watermark** (p. 1322)
	<<***extension***>> *(p. 213)* When the number of unacknowledged samples in the current send window of a reliable writer meets or exceeds this threshold, the **DDS_RELIABLE_WRITER_CACHE_CHANGED↩ _STATUS** (p. 545) is considered to have changed.
DDS_RtpsReliableWriterProtocol_t (p. 1320)	**virtual_heartbeat_period** (p. 1325)
	<<***extension***>> *(p. 213)* The period at which to send virtual heartbeats. Virtual heartbeats inform the reliable reader about the range of samples currently present, for each virtual GUID, in the reliable writer's queue.
DDS_RtpsReliableWriterProtocol_t (p. 1320)	**samples_per_virtual_heartbeat** (p. 1325)
	<<***extension***>> *(p. 213)* The number of samples that a reliable writer has to publish before sending a virtual heartbeat.
DDS_RtpsReliableWriterProtocol_t (p. 1320)	**max_heartbeat_retries** (p. 1326)

	<<***extension***>> *(p. 213)* The maximum number of *periodic* heartbeat retries before marking a remote reader as inactive.
DDS_RtpsReliableWriterProtocol_t (p. 1320)	**inactivate_nonprogressing_readers** (p. 1326)
	<<***extension***>> *(p. 213)* Whether to treat remote readers as inactive when their NACKs do not progress
DDS_RtpsReliableWriterProtocol_t (p. 1320)	**heartbeats_per_max_samples** (p. 1327)
	<<***extension***>> *(p. 213)* The number of piggyback heartbeats sent per max send window.
DDS_RtpsReliableWriterProtocol_t (p. 1320)	**min_nack_response_delay** (p. 1328)
	<<***extension***>> *(p. 213)* The minimum delay to respond to a NACK or NACK_FRAG.
DDS_RtpsReliableWriterProtocol_t (p. 1320)	**max_nack_response_delay** (p. 1328)
	<<***extension***>> *(p. 213)* The maximum delay to respond to a NACK or NACK_FRAG.
DDS_RtpsReliableWriterProtocol_t (p. 1320)	**nack_suppression_duration** (p. 1329)
	<<***extension***>> *(p. 213)* The duration for ignoring consecutive NACKs/NACK_FRAGs that may trigger redundant repairs.
DDS_RtpsReliableWriterProtocol_t (p. 1320)	**max_bytes_per_nack_response** (p. 1329)
	<<***extension***>> *(p. 213)* The maximum total message size when resending rejected samples or data fragments.
DDS_RtpsReliableWriterProtocol_t (p. 1320)	**disable_positive_acks_enable_adaptive_sample_keep_duration** (p. 1331)
	[DEPRECATED] (p. **384)** <<***extension***>> *(p. 213)* Enables dynamic adjustment of sample keep duration in response to congestion.
DDS_RtpsReliableWriterProtocol_t (p. 1320)	**disable_positive_acks_min_sample_keep_duration** (p. 1330)
	<<***extension***>> *(p. 213)* The minimum duration a sample is queued for ACK-disabled readers.
DDS_RtpsReliableWriterProtocol_t (p. 1320)	**disable_positive_acks_max_sample_keep_duration** (p. 1331)
	<<***extension***>> *(p. 213)* The maximum duration a sample is queued for ACK-disabled readers.
DDS_RtpsReliableWriterProtocol_t (p. 1320)	**disable_positive_acks_decrease_sample_keep_duration_factor** (p. 1332)

	[DEPRECATED] (p. **384**) <<***extension***>> *(p. 213)* Controls rate of contraction of dynamic sample keep duration.
DDS_RtpsReliableWriterProtocol_t (p. 1320)	**disable_positive_acks_increase_sample_keep_↩ duration_factor** (p. 1332)
	[DEPRECATED] (p. **384**) <<***extension***>> *(p. 213)* Controls rate of growth of dynamic sample keep duration.
DDS_RtpsReliableWriterProtocol_t (p. 1320)	**send_window_update_period** (p. 1334)
	<<***extension***>> *(p. 213)* Period in which send window may be dynamically changed
DDS_RtpsReliableWriterProtocol_t (p. 1320)	**send_window_increase_factor** (p. 1335)
	<<***extension***>> *(p. 213)* Increases send window size by this percentage when reacting dynamically to network conditions
DDS_RtpsReliableWriterProtocol_t (p. 1320)	**send_window_decrease_factor** (p. 1336)
	<<***extension***>> *(p. 213)* Decreases send window size by this percentage when reacting dynamically to network conditions
DDS_RtpsReliableWriterProtocol_t (p. 1320)	**enable_multicast_periodic_heartbeat** (p. 1337)
	<<***extension***>> *(p. 213)* Whether periodic heartbeat messages are sent over multicast
DDS_RtpsReliableWriterProtocol_t (p. 1320)	**multicast_resend_threshold** (p. 1337)
	<<***extension***>> *(p. 213)* The minimum number of requesting readers needed to trigger a multicast resend
DDS_RtpsReliableWriterProtocol_t (p. 1320)	**disable_repair_piggyback_heartbeat** (p. 1338)
	<<***extension***>> *(p. 213)* Prevents piggyback heartbeats from being sent with repair samples
DDS_BuiltinTopicReaderResourceLimits_t (p. 815)	**initial_samples** (p. 816)
	<<***extension***>> *(p. 213)* Initial number of samples.
DDS_BuiltinTopicReaderResourceLimits_t (p. 815)	**max_samples** (p. 816)
	<<***extension***>> *(p. 213)* Maximum number of samples.
DDS_BuiltinTopicReaderResourceLimits_t (p. 815)	**initial_infos** (p. 817)
	<<***extension***>> *(p. 213)* Initial number of sample infos.
DDS_BuiltinTopicReaderResourceLimits_t (p. 815)	**max_infos** (p. 817)
	<<***extension***>> *(p. 213)* Maximum number of sample infos.

DDS_BuiltinTopicReaderResourceLimits_t (p. 815)	**initial_outstanding_reads** (p. 817)
	<<***extension***>> *(p. 213)* The initial number of outstanding reads that have not called finish yet on the same built-in topic **DDSDataReader** (p. 1556).
DDS_BuiltinTopicReaderResourceLimits_t (p. 815)	**max_outstanding_reads** (p. 818)
	<<***extension***>> *(p. 213)* The maximum number of outstanding reads that have not called finish yet on the same built-in topic **DDSDataReader** (p. 1556).
DDS_BuiltinTopicReaderResourceLimits_t (p. 815)	**max_samples_per_read** (p. 818)
	<<***extension***>> *(p. 213)* Maximum number of samples that can be read/taken on a same built-in topic **DDSDataReader** (p. 1556).
DDS_BuiltinTopicReaderResourceLimits_t (p. 815)	**disable_fragmentation_support** (p. 818)
	[DEPRECATED] (p. 384) <<***extension***>> *(p. 213)* Determines whether the built-in topic **DDSDataReader** (p. 1556) can receive fragmented samples.
DDS_BuiltinTopicReaderResourceLimits_t (p. 815)	**max_fragmented_samples** (p. 819)
	<<***extension***>> *(p. 213)* The maximum number of samples for which the built-in topic **DDSDataReader** (p. 1556) may store fragments at a given point in time.
DDS_BuiltinTopicReaderResourceLimits_t (p. 815)	**initial_fragmented_samples** (p. 819)
	<<***extension***>> *(p. 213)* The initial number of samples for which a built-in topic **DDSDataReader** (p. 1556) may store fragments.
DDS_BuiltinTopicReaderResourceLimits_t (p. 815)	**max_fragmented_samples_per_remote_writer** (p. 820)
	<<***extension***>> *(p. 213)* The maximum number of samples per remote writer for which a built-in topic **DDSDataReader** (p. 1556) may store fragments.
DDS_BuiltinTopicReaderResourceLimits_t (p. 815)	**max_fragments_per_sample** (p. 820)
	<<***extension***>> *(p. 213)* Maximum number of fragments for a single sample.
DDS_BuiltinTopicReaderResourceLimits_t (p. 815)	**dynamically_allocate_fragmented_samples** (p. 821)
	<<***extension***>> *(p. 213)* Determines whether the built-in topic **DDSDataReader** (p. 1556) pre-allocates storage for storing fragmented samples.
DDS_DataTags (p. 889)	**tags** (p. 889)
	<<***spec***>> *(p. 209)* Sequence of data tags.
DDS_EntityNameQosPolicy (p. 1147)	**role_name** (p. 1148)

	<<***extension***>> *(p. 213)* The entity role name.
DDS_ServiceQosPolicy (p. 1358)	**kind** (p. 1359)
	<<***extension***>> *(p. 213)* The kind of service.
DDS_BatchQosPolicy (p. 810)	**max_flush_delay** (p. 812)
	<<***extension***>> *(p. 213)* The maximum flush delay.
DDS_BatchQosPolicy (p. 810)	**source_timestamp_resolution** (p. 812)
	<<***extension***>> *(p. 213)* Batch source timestamp resolution.
DDS_BatchQosPolicy (p. 810)	**thread_safe_write** (p. 813)
	<<***extension***>> *(p. 213)* Determines whether or not the write operation is thread safe.
DDS_ChannelSettings_t (p. 822)	**multicast_settings** (p. 822)
	<<***extension***>> *(p. 213)* A sequence of **DDS_TransportMulticastSettings_t** (p. 1417) used to configure the multicast addresses associated with a channel.
DDS_ChannelSettings_t (p. 822)	**filter_expression** (p. 822)
	<<***extension***>> *(p. 213)* A logical expression used to determine the data that will be published in the channel.
DDS_ChannelSettings_t (p. 822)	**priority** (p. 823)
	<<***extension***>> *(p. 213)* Publication priority
DDS_MultiChannelQosPolicy (p. 1214)	**channels** (p. 1215)
	<<***extension***>> *(p. 213)* A sequence of **DDS_ChannelSettings_t** (p. 822) used to configure the channels' properties. If the length of the sequence is zero, the QoS policy will be ignored.
DDS_MultiChannelQosPolicy (p. 1214)	**filter_name** (p. 1215)
	<<***extension***>> *(p. 213)* Name of the filter class used to describe the filter expressions of a MultiChannel DataWriter.
DDS_TopicQueryDispatchQosPolicy (p. 1404)	**enable** (p. 1405)
	<<***extension***>> *(p. 213)* Allows this writer to dispatch TopicQueries
DDS_TopicQueryDispatchQosPolicy (p. 1404)	**samples_per_period** (p. 1405)
	<<***extension***>> *(p. 213)* Sets the maximum number of samples to publish in each publication_period.

7.80 Mutable QoS

This page lists all QoS that are always mutable.

This page lists all QoS that are always mutable.

Mutable QoS can always be changed regardless of whether the entity has been enabled or not. QoS that are immutable after an entity has been enabled are listed on the **Immutable QoS** (p. 333) page.

Basic Settings

DDS_DeadlineQosPolicy (p. 928)	**period** (p. 930)
	*<<**basic**>> (p. 201) <<**spec**>> (p. 209)* Duration of the deadline period.
DDS_OwnershipStrengthQosPolicy (p. 1227)	**value** (p. 1228)
	*<<**basic**>> (p. 201) <<**spec**>> (p. 209)* The strength value used to arbitrate among multiple writers.
DDS_TimeBasedFilterQosPolicy (p. 1389)	**minimum_separation** (p. 1391)
	*<<**basic**>> (p. 201) <<**spec**>> (p. 209)* The minimum separation duration between subsequent samples.
DDS_PartitionQosPolicy (p. 1240)	**name** (p. 1242)
	*<<**basic**>> (p. 201) <<**spec**>> (p. 209)* A list of partition names
DDS_EntityFactoryQosPolicy (p. 1145)	**autoenable_created_entities** (p. 1147)
	*<<**basic**>> (p. 201) <<**spec**>> (p. 209)* Specifies whether the entity acting as a factory automatically enables the instances it creates.
DDS_LifespanQosPolicy (p. 1175)	**duration** (p. 1176)
	*<<**basic**>> (p. 201) <<**spec**>> (p. 209)* Maximum duration for the data's validity.
DDS_PropertyQosPolicy (p. 1260)	**value** (p. 1262)
	*<<**basic**>> (p. 201) <<**extension**>> (p. 213)* Sequence of properties.
DDS_LoggingQosPolicy (p. 1188)	**verbosity** (p. 1189)
	*<<**basic**>> (p. 201) <<**extension**>> (p. 213)* The verbosities at which RTI Connext diagnostic information is logged.
DDS_LoggingQosPolicy (p. 1188)	**category** (p. 1189)
	*<<**basic**>> (p. 201) <<**extension**>> (p. 213)* Categories of logged messages.

DDS_LoggingQosPolicy (p. 1188)	**output_file** (p. 1189)
	<<***basic***>> *(p. 201)* <<***extension***>> *(p. 213)* Specifies the file to which log messages will be redirected to.
DDS_LoggingQosPolicy (p. 1188)	**output_file_suffix** (p. 1190)
	<<***basic***>> *(p. 201)* <<***extension***>> *(p. 213)* Sets the file suffix when logging to a set of files
DDS_MonitoringEventDistributionSettings (p. 1197)	**publication_period** (p. 1200)
	<<***basic***>> *(p. 201)* <<***extension***>> *(p. 213)* Period at which the event metric thread publishes the event metrics that have changed since the last time they were published.
DDS_MonitoringPeriodicDistributionSettings (p. 1209)	**polling_period** (p. 1210)
	<<***basic***>> *(p. 201)* <<***extension***>> *(p. 213)* Period at which the periodic metric thread polls and publishes the periodic metrics.
DDS_MonitoringLoggingDistributionSettings (p. 1200)	**max_historical_logs** (p. 1202)
	<<***basic***>> *(p. 201)* <<***extension***>> *(p. 213)* The number of log messages that RTI Monitoring Library 2.0 will keep as history.
DDS_MonitoringLoggingDistributionSettings (p. 1200)	**publication_period** (p. 1203)
	<<***basic***>> *(p. 201)* <<***extension***>> *(p. 213)* Period at which the logging thread publishes log messages.
DDS_MonitoringDedicatedParticipantSettings (p. 1193)	**enable** (p. 1193)
	<<***basic***>> *(p. 201)* <<***extension***>> *(p. 213)* Enables the use of a dedicated **DDSDomainParticipant** (p. 1619) to distribute the RTI Connext application telemetry data.
DDS_MonitoringDedicatedParticipantSettings (p. 1193)	**domain_id** (p. 1194)
	<<***basic***>> *(p. 201)* <<***extension***>> *(p. 213)* The domain ID used in the creation of RTI Monitoring Library 2.0 **DDSDomainParticipant** (p. 1619).
DDS_MonitoringDedicatedParticipantSettings (p. 1193)	**collector_initial_peers** (p. 1194)
	<<***basic***>> *(p. 201)* <<***extension***>> *(p. 213)* Determines the initial list of peers that the discovery process will contact to send announcements about the presence of the **DDS_MonitoringDistributionSettings**↩ **::dedicated_participant** (p. 1196).

DDS_MonitoringLoggingForwardingSettings (p. 1204)	**middleware_forwarding_level** (p. 1204)
	*<<**basic**>> (p. 201) <<**extension**>> (p. 213)* Log messages with **NDDS_CONFIG_LOG_FACILITY_↩ MIDDLEWARE** (p. 727) and a log level more verbose than this value will not be distributed by RTI Monitoring Library 2.0.
DDS_MonitoringLoggingForwardingSettings (p. 1204)	**security_event_forwarding_level** (p. 1204)
	*<<**basic**>> (p. 201) <<**extension**>> (p. 213)* Log messages with **NDDS_CONFIG_LOG_FACILITY_↩ SECURITY_EVENT** (p. 727) and a log level more verbose than this value will not be distributed by RTI Monitoring Library 2.0.
DDS_MonitoringLoggingForwardingSettings (p. 1204)	**service_forwarding_level** (p. 1205)
	*<<**basic**>> (p. 201) <<**extension**>> (p. 213)* Log messages with **NDDS_CONFIG_LOG_FACILITY_↩ SERVICE** (p. 727) and a log level more verbose than this value will not be distributed by RTI Monitoring Library 2.0.
DDS_MonitoringLoggingForwardingSettings (p. 1204)	**user_forwarding_level** (p. 1205)
	*<<**basic**>> (p. 201) <<**extension**>> (p. 213)* Log messages with **NDDS_CONFIG_LOG_FACILITY_↩ USER** (p. 727) and a log level more verbose than this value will not be distributed by RTI Monitoring Library 2.0.
DDS_MonitoringTelemetryData (p. 1213)	**metrics** (p. 1213)
	*<<**basic**>> (p. 201) <<**extension**>> (p. 213)* Sequence of **DDS_MonitoringMetricSelection** (p. 1206) containing the event and periodic metrics that will be collected and distributed for a given set of observable resources.
DDS_MonitoringTelemetryData (p. 1213)	**logs** (p. 1213)
	*<<**basic**>> (p. 201) <<**extension**>> (p. 213)* **DDS_MonitoringLoggingForwardingSettings** (p. 1204) containing the **NDDS_Config_Syslog↩ Verbosity** (p. 727) levels that will be forwarded for the different **NDDS_Config_LogFacility** (p. 727).
DDS_MonitoringQosPolicy (p. 1210)	**enable** (p. 1211)
	*<<**basic**>> (p. 201) <<**extension**>> (p. 213)* Enables the collection and distribution of telemetry data for an RTI Connext application using RTI Monitoring Library 2.0.

DDS_MonitoringQosPolicy (p. 1210)	**application_name** (p. 1211)
	<<***basic***>> *(p. 201)* <<***extension***>> *(p. 213)* The name of the resource that represents this RTI Connext application.

Advanced Settings

DDS_UserDataQosPolicy (p. 1504)	**value** (p. 1505)
	<<***spec***>> *(p. 209)* a sequence of octets
DDS_GroupDataQosPolicy (p. 1160)	**value** (p. 1161)
	<<***spec***>> *(p. 209)* a sequence of octets
DDS_AvailabilityQosPolicy (p. 805)	**max_data_availability_waiting_time** (p. 808)
	<<***extension***>> *(p. 213)* Defines how much time to wait before delivering a sample to the application without having received some of the previous samples.
DDS_AvailabilityQosPolicy (p. 805)	**max_endpoint_availability_waiting_time** (p. 808)
	<<***extension***>> *(p. 213)* Defines how much time to wait to discover DataWriters providing samples for the same data source (virtual GUID).
DDS_AvailabilityQosPolicy (p. 805)	**required_matched_endpoint_groups** (p. 809)
	<<***extension***>> *(p. 213)* A sequence of endpoint groups.
DDS_LatencyBudgetQosPolicy (p. 1173)	**duration** (p. 1175)
	<<***spec***>> *(p. 209)* Duration of the maximum acceptable delay.
DDS_LoggingQosPolicy (p. 1188)	**print_format** (p. 1189)
	<<***extension***>> *(p. 213)* The format used to output RTI Connext diagnostic information.
DDS_LoggingQosPolicy (p. 1188)	**max_bytes_per_file** (p. 1190)
	<<***extension***>> *(p. 213)* Specifies the maximum number of bytes a single file can contain
DDS_LoggingQosPolicy (p. 1188)	**max_files** (p. 1191)
	<<***extension***>> *(p. 213)* Specifies the maximum number of files to create before overwritting the previous ones.
DDS_MonitoringEventDistributionSettings (p. 1197)	**concurrency_level** (p. 1198)
	<<***extension***>> *(p. 213)* Defines how concurrent the push of event metrics to RTI Monitoring Library 2.0 is.

DDS_MonitoringEventDistributionSettings (p. 1197)	**datawriter_qos_profile_name** (p. 1199)
	<<***extension***>> *(p. 213)* The fully qualified name of the profile used to configure the **DDSDataWriter** (p. 1589) that distributes event metrics.
DDS_MonitoringEventDistributionSettings (p. 1197)	**thread** (p. 1199)
	<<***extension***>> *(p. 213)* The settings of the event metric thread.
DDS_MonitoringPeriodicDistributionSettings (p. 1209)	**datawriter_qos_profile_name** (p. 1209)
	<<***extension***>> *(p. 213)* The fully qualified name of the profile used to configure the **DDSDataWriter** (p. 1589) that distributes periodic metrics.
DDS_MonitoringPeriodicDistributionSettings (p. 1209)	**thread** (p. 1209)
	<<***extension***>> *(p. 213)* The settings of the periodic metric thread.
DDS_MonitoringLoggingDistributionSettings (p. 1200)	**concurrency_level** (p. 1201)
	<<***extension***>> *(p. 213)* Defines how concurrent the push of log messages to RTI Monitoring Library 2.0 is.
DDS_MonitoringLoggingDistributionSettings (p. 1200)	**datawriter_qos_profile_name** (p. 1202)
	<<***extension***>> *(p. 213)* The fully qualified name of the profile used to configure the **DDSDataWriter** (p. 1589) that distributes log messages.
DDS_MonitoringLoggingDistributionSettings (p. 1200)	**thread** (p. 1203)
	<<***extension***>> *(p. 213)* The settings of the logging thread.
DDS_MonitoringDedicatedParticipantSettings (p. 1193)	**participant_qos_profile_name** (p. 1194)
	<<***extension***>> *(p. 213)* The fully qualified name of the profile used to configure the **DDSDomain↩ Participant** (p. 1619) that will be used to distribute telemetry data.
DDS_MonitoringDistributionSettings (p. 1195)	**dedicated_participant** (p. 1196)
	<<***extension***>> *(p. 213)* Configures the use of a dedicated **DDSDomainParticipant** (p. 1619) to distribute the RTI Connext application telemetry data.
DDS_MonitoringDistributionSettings (p. 1195)	**publisher_qos_profile_name** (p. 1196)

	<<***extension***>> *(p. 213)* The fully qualified name of the profile used to configure the Publishers that distribute telemetry data.
DDS_MonitoringDistributionSettings (p. 1195)	**event_settings** (p. 1196)
	<<***extension***>> *(p. 213)* Configures the distribution of event metrics.
DDS_MonitoringDistributionSettings (p. 1195)	**periodic_settings** (p. 1197)
	<<***extension***>> *(p. 213)* Configures the distribution of periodic metrics.
DDS_MonitoringDistributionSettings (p. 1195)	**logging_settings** (p. 1197)
	<<***extension***>> *(p. 213)* Configures the distribution of logging messages.
DDS_MonitoringMetricSelection (p. 1206)	**resource_selection** (p. 1206)
	<<***extension***>> *(p. 213)* An expression pattern that selects a subset of resources by matching the pattern to the resource names. **DDS_MonitoringMetric↩ Selection::enabled_metrics_selection** (p. 1207) and **DDS_MonitoringMetricSelection::disabled_↩ metrics_selection** (p. 1207) are applied to this subset of resources.
DDS_MonitoringMetricSelection (p. 1206)	**enabled_metrics_selection** (p. 1207)
	<<***extension***>> *(p. 213)* A sequence of POSIX fnmatch patterns that match the names of the metrics that should be collected and distributed for the observable resources selected by **DDS_MonitoringMetric↩ Selection::resource_selection** (p. 1206).
DDS_MonitoringMetricSelection (p. 1206)	**disabled_metrics_selection** (p. 1207)
	<<***extension***>> *(p. 213)* A sequence of POSIX fnmatch patterns that mach the names of the metrics that should not be collected and distributed for the observable resources selected by **DDS_MonitoringMetric↩ Selection::resource_selection** (p. 1206).
DDS_MonitoringQosPolicy (p. 1210)	**distribution_settings** (p. 1212)
	<<***extension***>> *(p. 213)* Configures the distribution of telemetry data.
DDS_MonitoringQosPolicy (p. 1210)	**telemetry_data** (p. 1212)
	<<***extension***>> *(p. 213)* Configures the telemetry data that will be distributed.

7.81 Other QoS Attributes

These attributes are characteristics of the QoS – for example if they are **mutable** (p. 369) or **immutable** (p. 333), or if they **apply per-instance** (p. 375).

Modules

- **Applied Per-Instance**

  These QoS are applied per-instance.
- **Discovery Metadata**

  These QoS are sent as part of one of the built-in discovery topics.
- **Matching**

  These QoS affect participant and endpoint matching.

7.81.1 Detailed Description

These attributes are characteristics of the QoS – for example if they are **mutable** (p. 369) or **immutable** (p. 333), or if they **apply per-instance** (p. 375).

7.82 Applied Per-Instance

These QoS are applied per-instance.

These QoS are applied per-instance.

For example, the **DDS_HistoryQosPolicy::depth** (p. 1164) configures the number of samples that are stored in the DataWriter and DataReader queues *per-instance*.

Basic Settings

DDS_DurabilityQosPolicy (p. 1005)	**writer_depth** (p. 1008)
	*<<**basic**>> (p. 201) <<**extension**>> (p. 213)* Indicates the number of samples per instance that a durable **DDSDataWriter** (p. 1589) will send to a late-joining **DDSDataReader** (p. 1556).
DDS_DeadlineQosPolicy (p. 928)	**period** (p. 930)
	*<<**basic**>> (p. 201) <<**spec**>> (p. 209)* Duration of the deadline period.
DDS_OwnershipQosPolicy (p. 1222)	**kind** (p. 1227)
	*<<**basic**>> (p. 201) <<**spec**>> (p. 209)* The kind of ownership
DDS_OwnershipStrengthQosPolicy (p. 1227)	**value** (p. 1228)

	<<***basic***>> *(p. 201)* <<***spec***>> *(p. 209)* The strength value used to arbitrate among multiple writers.
DDS_TimeBasedFilterQosPolicy (p. 1389)	**minimum_separation** (p. 1391)
	<<***basic***>> *(p. 201)* <<***spec***>> *(p. 209)* The minimum separation duration between subsequent samples.
DDS_DestinationOrderQosPolicy (p. 931)	**kind** (p. 933)
	<<***basic***>> *(p. 201)* <<***spec***>> *(p. 209)* Specifies the desired kind of destination order.
DDS_HistoryQosPolicy (p. 1162)	**kind** (p. 1164)
	<<***basic***>> *(p. 201)* <<***spec***>> *(p. 209)* Specifies the kind of history to be kept.
DDS_HistoryQosPolicy (p. 1162)	**depth** (p. 1164)
	<<***basic***>> *(p. 201)* <<***spec***>> *(p. 209)* Specifies the number of samples per instance to be kept, when the `kind` is **DDS_KEEP_LAST_HISTORY_QOS** (p. 604)

Advanced Settings

DDS_DurabilityServiceQosPolicy (p. 1009)	**history_kind** (p. 1010)
	<<***spec***>> *(p. 209)* The kind of history to apply in recouping durable data.
DDS_DurabilityServiceQosPolicy (p. 1009)	**history_depth** (p. 1011)
	<<***spec***>> *(p. 209)* Setting to use for the **DDS_DurabilityQosPolicy::writer_depth** (p. 1008) when recouping durable data.
DDS_PresentationQosPolicy (p. 1249)	**access_scope** (p. 1252)
	<<***spec***>> *(p. 209)* Determines the largest scope spanning the entities for which the order and coherency of changes can be preserved.
DDS_PresentationQosPolicy (p. 1249)	**ordered_access** (p. 1252)
	<<***spec***>> *(p. 209)* Specifies support for *ordered* access to the samples received at the subscription end. Controls whether ordered access is supported within the scope `access_scope`.
DDS_DestinationOrderQosPolicy (p. 931)	**scope** (p. 933)
	<<***spec***>> *(p. 209)* Specifies the desired scope of the source destination order.
DDS_DestinationOrderQosPolicy (p. 931)	**source_timestamp_tolerance** (p. 933)
	<<***extension***>> *(p. 213)* Allowed tolerance between source timestamps of consecutive samples.

7.83 Discovery Metadata

These QoS are sent as part of one of the built-in discovery topics.

These QoS are sent as part of one of the built-in discovery topics.

See also

DDS_ParticipantBuiltinTopicData (p. 1229)

DDS_PublicationBuiltinTopicData (p. 1264)

DDS_SubscriptionBuiltinTopicData (p. 1370)

Basic Settings

DDS_DurabilityQosPolicy (p. 1005)	**kind** (p. 1007)
	*<<**basic**>> (p. 201) <<**spec**>> (p. 209)* The kind of durability.
DDS_DataRepresentationQosPolicy (p. 886)	**compression_settings** (p. 888)
	*<<**basic**>> (p. 201) <<**extension**>> (p. 213)* Structure that contains the compression settings.
DDS_DeadlineQosPolicy (p. 928)	**period** (p. 930)
	*<<**basic**>> (p. 201) <<**spec**>> (p. 209)* Duration of the deadline period.
DDS_OwnershipQosPolicy (p. 1222)	**kind** (p. 1227)
	*<<**basic**>> (p. 201) <<**spec**>> (p. 209)* The kind of ownership
DDS_OwnershipStrengthQosPolicy (p. 1227)	**value** (p. 1228)
	*<<**basic**>> (p. 201) <<**spec**>> (p. 209)* The strength value used to arbitrate among multiple writers.
DDS_LivelinessQosPolicy (p. 1180)	**kind** (p. 1182)
	*<<**basic**>> (p. 201) <<**spec**>> (p. 209)* The kind of liveliness desired.
DDS_LivelinessQosPolicy (p. 1180)	**lease_duration** (p. 1182)
	*<<**basic**>> (p. 201) <<**spec**>> (p. 209)* The duration within which a **DDSDataWriter** (p. 1589) must be asserted, or else it is assumed to be not alive.
DDS_TimeBasedFilterQosPolicy (p. 1389)	**minimum_separation** (p. 1391)
	*<<**basic**>> (p. 201) <<**spec**>> (p. 209)* The minimum separation duration between subsequent samples.
DDS_PartitionQosPolicy (p. 1240)	**name** (p. 1242)

	*<<**basic**>> (p. 201) <<**spec**>> (p. 209)* A list of partition names
DDS_ReliabilityQosPolicy (p. 1295)	**kind** (p. 1298)
	*<<**basic**>> (p. 201) <<**spec**>> (p. 209)* Kind of reliability.
DDS_ReliabilityQosPolicy (p. 1295)	**max_blocking_time** (p. 1298)
	*<<**basic**>> (p. 201) <<**spec**>> (p. 209)* The maximum time a DataWriter can block on a write() call.
DDS_DestinationOrderQosPolicy (p. 931)	**kind** (p. 933)
	*<<**basic**>> (p. 201) <<**spec**>> (p. 209)* Specifies the desired kind of destination order.
DDS_LifespanQosPolicy (p. 1175)	**duration** (p. 1176)
	*<<**basic**>> (p. 201) <<**spec**>> (p. 209)* Maximum duration for the data's validity.
DDS_TransportMulticastQosPolicy (p. 1415)	**value** (p. 1416)
	*<<**basic**>> (p. 201) <<**extension**>> (p. 213)* A sequence of multicast communications settings.
DDS_DiscoveryQosPolicy (p. 960)	**multicast_receive_addresses** (p. 962)
	*<<**basic**>> (p. 201) <<**extension**>> (p. 213)* Specifies the multicast group addresses on which discovery-related **meta-traffic** can be received by the DomainParticipant.
DDS_DiscoveryConfigQosPolicy (p. 934)	**participant_liveliness_lease_duration** (p. 937)
	*<<**basic**>> (p. 201) <<**extension**>> (p. 213)* The liveliness lease duration for the participant.
DDS_PropertyQosPolicy (p. 1260)	**value** (p. 1262)
	*<<**basic**>> (p. 201) <<**extension**>> (p. 213)* Sequence of properties.
DDS_EntityNameQosPolicy (p. 1147)	**name** (p. 1148)
	*<<**basic**>> (p. 201) <<**extension**>> (p. 213)* The name of the entity.

Advanced Settings

DDS_UserDataQosPolicy (p. 1504)	**value** (p. 1505)
	*<<**spec**>> (p. 209)* a sequence of octets
DDS_GroupDataQosPolicy (p. 1160)	**value** (p. 1161)
	*<<**spec**>> (p. 209)* a sequence of octets

DDS_DurabilityQosPolicy (p. 1005)	**direct_communication** (p. 1007)
	<<***extension***>> *(p. 213)* Indicates whether or not a TRANSIENT or PERSISTENT **DDSDataReader** (p. 1556) should receive samples directly from a TRANSIENT or PERSISTENT **DDSDataWriter** (p. 1589)
DDS_DurabilityServiceQosPolicy (p. 1009)	**service_cleanup_delay** (p. 1010)
	<<***spec***>> *(p. 209)* Controls when the service is able to remove all information regarding a data instances.
DDS_DurabilityServiceQosPolicy (p. 1009)	**history_kind** (p. 1010)
	<<***spec***>> *(p. 209)* The kind of history to apply in recouping durable data.
DDS_DurabilityServiceQosPolicy (p. 1009)	**history_depth** (p. 1011)
	<<***spec***>> *(p. 209)* Setting to use for the **DDS_DurabilityQosPolicy::writer_depth** (p. 1008) when recouping durable data.
DDS_DurabilityServiceQosPolicy (p. 1009)	**max_samples** (p. 1011)
	<<***spec***>> *(p. 209)* Part of resource limits QoS policy to apply when feeding a late joiner.
DDS_DurabilityServiceQosPolicy (p. 1009)	**max_instances** (p. 1011)
	<<***spec***>> *(p. 209)* Part of resource limits QoS policy to apply when feeding a late joiner.
DDS_DurabilityServiceQosPolicy (p. 1009)	**max_samples_per_instance** (p. 1012)
	<<***spec***>> *(p. 209)* Part of resource limits QoS policy to apply when feeding a late joiner.
DDS_CompressionSettings_t (p. 826)	**compression_ids** (p. 826)
	<<***extension***>> *(p. 213)* Mask that represents the compression algorithms enabled.
DDS_DataRepresentationQosPolicy (p. 886)	**value** (p. 888)
	<<***spec***>> *(p. 209)* Sequence of representation identifiers.
DDS_PresentationQosPolicy (p. 1249)	**access_scope** (p. 1252)
	<<***spec***>> *(p. 209)* Determines the largest scope spanning the entities for which the order and coherency of changes can be preserved.
DDS_PresentationQosPolicy (p. 1249)	**coherent_access** (p. 1252)
	<<***spec***>> *(p. 209)* Specifies support for *coherent* access. Controls whether coherent access is supported within the scope `access_scope`.
DDS_PresentationQosPolicy (p. 1249)	**ordered_access** (p. 1252)

	<<***spec***>> *(p. 209)* Specifies support for *ordered* access to the samples received at the subscription end. Controls whether ordered access is supported within the scope `access_scope`.
DDS_LatencyBudgetQosPolicy (p. 1173)	**duration** (p. 1175)
	<<***spec***>> *(p. 209)* Duration of the maximum acceptable delay.
DDS_ReliabilityQosPolicy (p. 1295)	**acknowledgment_kind** (p. 1298)
	<<***extension***>> *(p. 213)* Kind of reliable acknowledgment
DDS_ReliabilityQosPolicy (p. 1295)	**instance_state_consistency_kind** (p. 1299)
	<<***extension***>> *(p. 213)* Whether instance state consistency is enabled
DDS_ReliabilityQosPolicy (p. 1295)	**exact_reliability_kind_match** (p. 1299)
	<<***extension***>> *(p. 213)* Whether an exact reliability kind match is required for endpoint matching.
DDS_TypeConsistencyEnforcementQosPolicy (p. 1493)	**kind** (p. 1494)
	<<***spec***>> *(p. 209)* Type consistency kind.
DDS_TypeConsistencyEnforcementQosPolicy (p. 1493)	**ignore_sequence_bounds** (p. 1495)
	<<***spec***>> *(p. 209)* Controls whether sequence bounds are taken into consideration for type assignability.
DDS_TypeConsistencyEnforcementQosPolicy (p. 1493)	**ignore_string_bounds** (p. 1495)
	<<***spec***>> *(p. 209)* Controls whether string bounds are taken into consideration for type assignability.
DDS_TypeConsistencyEnforcementQosPolicy (p. 1493)	**ignore_member_names** (p. 1495)
	<<***spec***>> *(p. 209)* Controls whether member names are taken into consideration for type assignability.
DDS_TypeConsistencyEnforcementQosPolicy (p. 1493)	**prevent_type_widening** (p. 1496)
	<<***spec***>> *(p. 209)* Controls whether type widening is allowed.
DDS_TypeConsistencyEnforcementQosPolicy (p. 1493)	**force_type_validation** (p. 1496)

	<<***spec***>> *(p. 209)* Controls whether type information must be available in order to complete matching between a **DDSDataWriter** (p. 1589) and a **DDSData↩ Reader** (p. 1556).
DDS_TypeConsistencyEnforcementQosPolicy (p. 1493)	**ignore_enum_literal_names** (p. 1496)
	<<***spec***>> *(p. 209)* Controls whether enumeration constant names are taken into consideration for type assignability.
DDS_TransportUnicastQosPolicy (p. 1422)	**value** (p. 1424)
	<<***extension***>> *(p. 213)* A sequence of unicast communication settings.
DDS_DataReaderProtocolQosPolicy (p. 843)	**virtual_guid** (p. 844)
	<<***extension***>> *(p. 213)* The virtual GUID (Global Unique Identifier).
DDS_DataReaderProtocolQosPolicy (p. 843)	**expects_inline_qos** (p. 844)
	[DEPRECATED] (p. 384) <<***extension***>> *(p. 213)* Specifies whether this DataReader expects inline QoS with every sample.
DDS_DataReaderProtocolQosPolicy (p. 843)	**disable_positive_acks** (p. 845)
	<<***extension***>> *(p. 213)* Whether the reader sends positive acknowledgements to writers.
DDS_DataWriterProtocolQosPolicy (p. 892)	**virtual_guid** (p. 893)
	<<***extension***>> *(p. 213)* The virtual GUID (Global Unique Identifier).
DDS_DataWriterProtocolQosPolicy (p. 892)	**disable_positive_acks** (p. 894)
	<<***extension***>> *(p. 213)* Controls whether or not the writer expects positive acknowledgements from matching readers.
DDS_DataTags (p. 889)	**tags** (p. 889)
	<<***spec***>> *(p. 209)* Sequence of data tags.
DDS_EntityNameQosPolicy (p. 1147)	**role_name** (p. 1148)
	<<***extension***>> *(p. 213)* The entity role name.
DDS_ServiceQosPolicy (p. 1358)	**kind** (p. 1359)
	<<***extension***>> *(p. 213)* The kind of service.
DDS_MultiChannelQosPolicy (p. 1214)	**channels** (p. 1215)

	*<<**extension**>> (p. 213)* A sequence of **DDS_ChannelSettings_t** (p. 822) used to configure the channels' properties. If the length of the sequence is zero, the QoS policy will be ignored.
DDS_MultiChannelQosPolicy (p. 1214)	**filter_name** (p. 1215)
	*<<**extension**>> (p. 213)* Name of the filter class used to describe the filter expressions of a MultiChannel DataWriter.
DDS_TopicQueryDispatchQosPolicy (p. 1404)	**enable** (p. 1405)
	*<<**extension**>> (p. 213)* Allows this writer to dispatch TopicQueries

7.84 Matching

These QoS affect participant and endpoint matching.

These QoS affect participant and endpoint matching.

See the pages of each individual QoS for a desription of how the QoS must be set in order for two applications to be considered matching. Look for information about compatibility or request-offered (**RxO** (p. 556)) semantics.

Basic Settings

DDS_DurabilityQosPolicy (p. 1005)	**kind** (p. 1007)
	*<<**basic**>> (p. 201) <<**spec**>> (p. 209)* The kind of durability.
DDS_DataRepresentationQosPolicy (p. 886)	**compression_settings** (p. 888)
	*<<**basic**>> (p. 201) <<**extension**>> (p. 213)* Structure that contains the compression settings.
DDS_DeadlineQosPolicy (p. 928)	**period** (p. 930)
	*<<**basic**>> (p. 201) <<**spec**>> (p. 209)* Duration of the deadline period.
DDS_OwnershipQosPolicy (p. 1222)	**kind** (p. 1227)
	*<<**basic**>> (p. 201) <<**spec**>> (p. 209)* The kind of ownership
DDS_LivelinessQosPolicy (p. 1180)	**kind** (p. 1182)
	*<<**basic**>> (p. 201) <<**spec**>> (p. 209)* The kind of liveliness desired.
DDS_LivelinessQosPolicy (p. 1180)	**lease_duration** (p. 1182)
	*<<**basic**>> (p. 201) <<**spec**>> (p. 209)* The duration within which a **DDSDataWriter** (p. 1589) must be asserted, or else it is assumed to be not alive.

DDS_PartitionQosPolicy (p. 1240)	**name** (p. 1242)
	<<***basic***>> *(p. 201)* <<***spec***>> *(p. 209)* A list of partition names
DDS_ReliabilityQosPolicy (p. 1295)	**kind** (p. 1298)
	<<***basic***>> *(p. 201)* <<***spec***>> *(p. 209)* Kind of reliability.
DDS_DestinationOrderQosPolicy (p. 931)	**kind** (p. 933)
	<<***basic***>> *(p. 201)* <<***spec***>> *(p. 209)* Specifies the desired kind of destination order.

Advanced Settings

DDS_CompressionSettings_t (p. 826)	**compression_ids** (p. 826)
	<<***extension***>> *(p. 213)* Mask that represents the compression algorithms enabled.
DDS_DataRepresentationQosPolicy (p. 886)	**value** (p. 888)
	<<***spec***>> *(p. 209)* Sequence of representation identifiers.
DDS_PresentationQosPolicy (p. 1249)	**access_scope** (p. 1252)
	<<***spec***>> *(p. 209)* Determines the largest scope spanning the entities for which the order and coherency of changes can be preserved.
DDS_PresentationQosPolicy (p. 1249)	**coherent_access** (p. 1252)
	<<***spec***>> *(p. 209)* Specifies support for *coherent* access. Controls whether coherent access is supported within the scope `access_scope`.
DDS_PresentationQosPolicy (p. 1249)	**ordered_access** (p. 1252)
	<<***spec***>> *(p. 209)* Specifies support for *ordered* access to the samples received at the subscription end. Controls whether ordered access is supported within the scope `access_scope`.
DDS_LatencyBudgetQosPolicy (p. 1173)	**duration** (p. 1175)
	<<***spec***>> *(p. 209)* Duration of the maximum acceptable delay.
DDS_ReliabilityQosPolicy (p. 1295)	**acknowledgment_kind** (p. 1298)
	<<***extension***>> *(p. 213)* Kind of reliable acknowledgment
DDS_ReliabilityQosPolicy (p. 1295)	**instance_state_consistency_kind** (p. 1299)

	<<***extension***>> *(p. 213)* Whether instance state consistency is enabled
DDS_ReliabilityQosPolicy (p. 1295)	**exact_reliability_kind_match** (p. 1299)
	<<***extension***>> *(p. 213)* Whether an exact reliability kind match is required for endpoint matching.
DDS_TypeConsistencyEnforcementQosPolicy (p. 1493)	**kind** (p. 1494)
	<<***spec***>> *(p. 209)* Type consistency kind.
DDS_TypeConsistencyEnforcementQosPolicy (p. 1493)	**ignore_sequence_bounds** (p. 1495)
	<<***spec***>> *(p. 209)* Controls whether sequence bounds are taken into consideration for type assignability.
DDS_TypeConsistencyEnforcementQosPolicy (p. 1493)	**ignore_string_bounds** (p. 1495)
	<<***spec***>> *(p. 209)* Controls whether string bounds are taken into consideration for type assignability.
DDS_TypeConsistencyEnforcementQosPolicy (p. 1493)	**ignore_member_names** (p. 1495)
	<<***spec***>> *(p. 209)* Controls whether member names are taken into consideration for type assignability.
DDS_TypeConsistencyEnforcementQosPolicy (p. 1493)	**prevent_type_widening** (p. 1496)
	<<***spec***>> *(p. 209)* Controls whether type widening is allowed.
DDS_TypeConsistencyEnforcementQosPolicy (p. 1493)	**force_type_validation** (p. 1496)
	<<***spec***>> *(p. 209)* Controls whether type information must be available in order to complete matching between a **DDSDataWriter** (p. 1589) and a **DDSData↩ Reader** (p. 1556).
DDS_TypeConsistencyEnforcementQosPolicy (p. 1493)	**ignore_enum_literal_names** (p. 1496)
	<<***spec***>> *(p. 209)* Controls whether enumeration constant names are taken into consideration for type assignability.

7.85 Deprecated QoS

Deprecated QoS.

Deprecated QoS.

This page lists the QoS that are deprecated. Deprecated QoS are still supported, but RTI will not support them in future releases.

DDS_ResourceLimitsQosPolicy (p. 1308)	**instance_hash_buckets** (p. 1313)
	[DEPRECATED] (p. **384**) <<***extension***>> *(p. 213)* Number of hash buckets for instances.
DDS_TypeSupportQosPolicy (p. 1499)	**plugin_data** (p. 1500)
	[DEPRECATED] (p. **384**) <<***extension***>> *(p. 213)* Value to pass into the type plugin's de-/serialization function
DDS_TypeSupportQosPolicy (p. 1499)	**cdr_padding_kind** (p. 1500)
	[DEPRECATED] (p. **384**) <<***extension***>> *(p. 213)* Determines whether or not the padding bytes will be set to zero during CDR serialization.
DDS_AsynchronousPublisherQosPolicy (p. 798)	**disable_asynchronous_write** (p. 800)
	[DEPRECATED] (p. **384**) <<***extension***>> *(p. 213)* Disable asynchronous publishing.
DDS_AsynchronousPublisherQosPolicy (p. 798)	**disable_asynchronous_batch** (p. 800)
	[DEPRECATED] (p. **384**) <<***extension***>> *(p. 213)* Disable asynchronous batch flushing.
DDS_AsynchronousPublisherQosPolicy (p. 798)	**disable_topic_query_publication** (p. 801)
	[DEPRECATED] (p. **384**) <<***extension***>> *(p. 213)* Disable topic query publication.
DDS_DiscoveryQosPolicy (p. 960)	**enable_endpoint_discovery** (p. 964)
	[DEPRECATED] (p. **384**) <<***extension***>> *(p. 213)* Whether to automatically enable endpoint discovery for all the remote participants.
DDS_DataReaderResourceLimitsQosPolicy (p. 867)	**disable_fragmentation_support** (p. 874)
	[DEPRECATED] (p. **384**) <<***extension***>> *(p. 213)* Determines whether the **DDSDataReader** (p. 1556) can receive fragmented samples.
DDS_DataWriterResourceLimitsQosPolicy (p. 917)	**initial_concurrent_blocking_threads** (p. 919)
	[DEPRECATED] (p. **384**) <<***extension***>> *(p. 213)* The initial number of threads that are allowed to concurrently block on write call on the same **DDSDataWriter** (p. 1589).
DDS_DataWriterResourceLimitsQosPolicy (p. 917)	**max_concurrent_blocking_threads** (p. 919)

	[DEPRECATED] (p. **384**) <<***extension***>> *(p. 213)* The maximum number of threads that are allowed to concurrently block on write call on the same **DDSDataWriter** (p. 1589).
DDS_DataWriterResourceLimitsQosPolicy (p. 917)	**autoregister_instances** (p. 922)
	[DEPRECATED] (p. **384**) <<***extension***>> *(p. 213)* Whether or not to automatically register new instances
DDS_DataReaderProtocolQosPolicy (p. 843)	**expects_inline_qos** (p. 844)
	[DEPRECATED] (p. **384**) <<***extension***>> *(p. 213)* Specifies whether this DataReader expects inline QoS with every sample.
DDS_SystemResourceLimitsQosPolicy (p. 1382)	**max_objects_per_thread** (p. 1383)
	[DEPRECATED] (p. **384**) <<***extension***>> *(p. 213)* The maximum number of objects that can be stored per thread for a **DDSDomainParticipantFactory** (p. 1694).
DDS_DomainParticipantResourceLimitsQosPolicy (p. 977)	**local_writer_hash_buckets** (p. 986)
	[DEPRECATED] (p. **384**) <<***extension***>> *(p. 213)* Hash_Buckets settings applied to local DataWriters.
DDS_DomainParticipantResourceLimitsQosPolicy (p. 977)	**local_reader_hash_buckets** (p. 986)
	[DEPRECATED] (p. **384**) <<***extension***>> *(p. 213)* Number of hash buckets for local DataReaders.
DDS_DomainParticipantResourceLimitsQosPolicy (p. 977)	**local_publisher_hash_buckets** (p. 987)
	[DEPRECATED] (p. **384**) <<***extension***>> *(p. 213)* Number of hash buckets for local Publisher.
DDS_DomainParticipantResourceLimitsQosPolicy (p. 977)	**local_subscriber_hash_buckets** (p. 987)
	[DEPRECATED] (p. **384**) <<***extension***>> *(p. 213)* Number of hash buckets for local Subscriber.
DDS_DomainParticipantResourceLimitsQosPolicy (p. 977)	**local_topic_hash_buckets** (p. 987)
	[DEPRECATED] (p. **384**) <<***extension***>> *(p. 213)* Number of hash buckets for local Topic.
DDS_DomainParticipantResourceLimitsQosPolicy (p. 977)	**remote_writer_hash_buckets** (p. 988)
	[DEPRECATED] (p. **384**) <<***extension***>> *(p. 213)* Number of hash buckets for remote DataWriters.
DDS_DomainParticipantResourceLimitsQosPolicy (p. 977)	**remote_reader_hash_buckets** (p. 988)

	[DEPRECATED] (p. **384**) <<***extension***>> *(p. 213)* Number of hash buckets for remote DataReaders.
DDS_DomainParticipantResourceLimitsQosPolicy (p. 977)	**remote_participant_hash_buckets** (p. 988)
	[DEPRECATED] (p. **384**) <<***extension***>> *(p. 213)* Number of hash buckets for remote DomainParticipants.
DDS_DomainParticipantResourceLimitsQosPolicy (p. 977)	**matching_writer_reader_pair_hash_buckets** (p. 989)
	[DEPRECATED] (p. **384**) <<***extension***>> *(p. 213)* Number of hash buckets for matching local writer and remote/local reader pairs.
DDS_DomainParticipantResourceLimitsQosPolicy (p. 977)	**matching_reader_writer_pair_hash_buckets** (p. 989)
	[DEPRECATED] (p. **384**) <<***extension***>> *(p. 213)* Number of hash buckets for matching local reader and remote/local writer pairs.
DDS_DomainParticipantResourceLimitsQosPolicy (p. 977)	**ignored_entity_hash_buckets** (p. 989)
	[DEPRECATED] (p. **384**) <<***extension***>> *(p. 213)* Number of hash buckets for ignored entities.
DDS_DomainParticipantResourceLimitsQosPolicy (p. 977)	**content_filtered_topic_hash_buckets** (p. 990)
	[DEPRECATED] (p. **384**) <<***extension***>> *(p. 213)* Number of hash buckets for content filtered topics.
DDS_DomainParticipantResourceLimitsQosPolicy (p. 977)	**content_filter_hash_buckets** (p. 990)
	[DEPRECATED] (p. **384**) <<***extension***>> *(p. 213)* Number of hash buckets for content filters.
DDS_DomainParticipantResourceLimitsQosPolicy (p. 977)	**flow_controller_hash_buckets** (p. 990)
	[DEPRECATED] (p. **384**) <<***extension***>> *(p. 213)* Number of hash buckets for flow controllers.
DDS_DomainParticipantResourceLimitsQosPolicy (p. 977)	**type_code_max_serialized_length** (p. 994)
	[DEPRECATED] (p. **384**) <<***extension***>> *(p. 213)* Maximum size of serialized string for type code.
DDS_DomainParticipantResourceLimitsQosPolicy (p. 977)	**ignored_entity_replacement_kind** (p. 1001)

	[DEPRECATED] (p. **384**) <<***extension***>> *(p. 213)* Replacement policy for the ignored entities. It sets what entity can be replaced when resource limits set in **DDS_DomainParticipantResourceLimitsQosPolicy**↩ **::ignored_entity_allocation** (p. 984) are reached.
DDS_DomainParticipantResourceLimitsQosPolicy (p. 977)	**remote_topic_query_hash_buckets** (p. 1002)
	[DEPRECATED] (p. **384**) <<***extension***>> *(p. 213)* Number of hash buckets for remote TopicQueries.
DDS_DatabaseQosPolicy (p. 831)	**shutdown_timeout** (p. 832)
	[DEPRECATED] (p. **384**) <<***extension***>> *(p. 213)* The maximum wait time during a shutdown.
DDS_DatabaseQosPolicy (p. 831)	**cleanup_period** (p. 833)
	[DEPRECATED] (p. **384**) <<***extension***>> *(p. 213)* The period at which the database thread wakes up to remove deleted records.
DDS_DatabaseQosPolicy (p. 831)	**shutdown_cleanup_period** (p. 833)
	[DEPRECATED] (p. **384**) <<***extension***>> *(p. 213)* The clean-up period used during database shut-down.
DDS_DatabaseQosPolicy (p. 831)	**max_skiplist_level** (p. 834)
	[DEPRECATED] (p. **384**) <<***extension***>> *(p. 213)* The maximum level of the skiplist.
DDS_ReceiverPoolQosPolicy (p. 1293)	**buffer_alignment** (p. 1295)
	[DEPRECATED] (p. **384**) <<***extension***>> *(p. 213)* The receive buffer alignment.
DDS_DiscoveryConfigQosPolicy (p. 934)	**remote_participant_purge_kind** (p. 938)
	[DEPRECATED] (p. **384**) <<***extension***>> *(p. 213)* The participant's behavior for maintaining knowledge of remote participants (and their contained entities) with which discovery communication has been lost.
DDS_DiscoveryConfigQosPolicy (p. 934)	**max_liveliness_loss_detection_period** (p. 938)
	[DEPRECATED] (p. **384**) <<***extension***>> *(p. 213)* The maximum amount of time between when a remote entity stops maintaining its liveliness and when the matched local entity realizes that fact.
DDS_DiscoveryConfigQosPolicy (p. 934)	**ignore_default_domain_announcements** (p. 950)
	[DEPRECATED] (p. **384**) <<***extension***>> *(p. 213)* Used to ignore the announcements received by a participant on the default domain 0 corresponding to participants running on domains IDs other than 0.
DDS_DiscoveryConfigQosPolicy (p. 934)	**participant_message_reader_reliability_kind** (p. 946)

	[DEPRECATED] (p. **384)** <<***extension***>> *(*p. *213)* Reliability policy for a built-in participant message reader.
DDS_RtpsReliableWriterProtocol_t (p. 1320)	**disable_positive_acks_enable_adaptive_sample_keep_duration** (p. 1331)
	[DEPRECATED] (p. **384)** <<***extension***>> *(*p. *213)* Enables dynamic adjustment of sample keep duration in response to congestion.
DDS_RtpsReliableWriterProtocol_t (p. 1320)	**disable_positive_acks_decrease_sample_keep_duration_factor** (p. 1332)
	[DEPRECATED] (p. **384)** <<***extension***>> *(*p. *213)* Controls rate of contraction of dynamic sample keep duration.
DDS_RtpsReliableWriterProtocol_t (p. 1320)	**disable_positive_acks_increase_sample_keep_duration_factor** (p. 1332)
	[DEPRECATED] (p. **384)** <<***extension***>> *(*p. *213)* Controls rate of growth of dynamic sample keep duration.
DDS_BuiltinTopicReaderResourceLimits_t (p. 815)	**disable_fragmentation_support** (p. 818)
	[DEPRECATED] (p. **384)** <<***extension***>> *(*p. *213)* Determines whether the built-in topic **DDSDataReader** (p. 1556) can receive fragmented samples.

7.86 System Properties

System Properties.

System Properties.

RTI Connext uses the **DDS_PropertyQosPolicy** (p. 1260) of a DomainParticipant to maintain a set of properties that provide system information such as hostname.

Unless the default **DDS_DomainParticipantQos** (p. 971) value is overwritten, the system properties are automatically set in the **DDS_DomainParticipantQos** (p. 971) obtained by calling the method **DDSDomainParticipantFactory::get_default_participant_qos** (p. 1700) or using the constant **DDS_PARTICIPANT_QOS_DEFAULT** (p. 50).

System properties are also automatically set in the **DDS_DomainParticipantQos** (p. 971) loaded from an XML QoS profile unless you disable property inheritance using the attribute **inherit** in the XML tag <**property**>.

By default, the system properties are propagated to other DomainParticipants in the system and can be accessed through **DDS_ParticipantBuiltinTopicData::property** (p. 1230).

You can disable the propagation of individual properties by setting the flag **DDS_Property_t::propagate** (p. 1260) to **DDS_BOOLEAN_FALSE** (p. 513) or by removing the property using the method **DDSPropertyQosPolicyHelper::remove_property** (p. 625).

The number of system properties set on the **DDS_DomainParticipantQos** (p. 971) is platform specific.

7.86.1 System Properties List

The following table lists the supported system properties. For more information, see `System Properties`, in the Core Libraries User's Manual.

Property Name	Description
dds.sys_info.hostname	Name of the host where the process is running
dds.sys_info.process_id	Process ID
dds.sys_info.username	Name of the user running the process
dds.sys_info.execution_timestamp	Time when the execution started
dds.sys_info.creation_timestamp	Time when the executable was created
dds.sys_info.executable_filepath	Name and full path of the executable
dds.sys_info.target	Architecture for which the library was compiled

7.86.2 System Resource Consideration

System properties are affected by the resource limits **DDS_DomainParticipantResourceLimitsQosPolicy↩ ::participant_property_list_max_length** (p. 998) and **DDS_DomainParticipantResourceLimitsQosPolicy↩ ::participant_property_string_max_length** (p. 998).

7.87 Configuring QoS Profiles with XML

APIs related to XML QoS Profiles.

APIs related to XML QoS Profiles.

7.87.1 Loading QoS Profiles from XML Resources

A 'QoS profile' is a group of QoS settings, specified in XML format. By using QoS profiles, you can change QoS settings without recompiling the application.

The Qos profiles are loaded the first time any of the following operations are called:

- **DDSDomainParticipantFactory::create_participant** (p. 1710)
- **DDSDomainParticipantFactory::create_participant_with_profile** (p. 1711)
- **DDSDomainParticipantFactory::set_default_participant_qos_with_profile** (p. 1699)
- **DDSDomainParticipantFactory::get_default_participant_qos** (p. 1700)
- **DDSDomainParticipantFactory::set_default_library** (p. 1701)

- **DDSDomainParticipantFactory::set_default_profile** (p. 1702)
- **DDSDomainParticipantFactory::get_participant_qos_from_profile** (p. 1704)
- **DDSDomainParticipantFactory::get_topic_qos_from_profile** (p. 1708)
- **DDSDomainParticipantFactory::get_topic_qos_from_profile_w_topic_name** (p. 1709)
- **DDSDomainParticipantFactory::get_publisher_qos_from_profile** (p. 1705)
- **DDSDomainParticipantFactory::get_subscriber_qos_from_profile** (p. 1705)
- **DDSDomainParticipantFactory::get_datawriter_qos_from_profile** (p. 1706)
- **DDSDomainParticipantFactory::get_datawriter_qos_from_profile_w_topic_name** (p. 1706)
- **DDSDomainParticipantFactory::get_datareader_qos_from_profile** (p. 1707)
- **DDSDomainParticipantFactory::get_datareader_qos_from_profile_w_topic_name** (p. 1708)
- **DDSDomainParticipantFactory::get_qos_profile_libraries** (p. 1710)
- **DDSDomainParticipantFactory::get_qos_profiles** (p. 1710)
- **DDSDomainParticipantFactory::load_profiles** (p. 1715)

The QoS profiles are reloaded replacing previously loaded profiles when the following operations are called:

- **DDSDomainParticipantFactory::set_qos** (p. 1714)
- **DDSDomainParticipantFactory::reload_profiles** (p. 1715)

The **DDSDomainParticipantFactory::unload_profiles** (p. 1716) operation will free the resources associated with the XML QoS profiles.

There are five ways to configure the XML resources (listed by load order):

- The file NDDS_QOS_PROFILES.xml in $NDDSHOME/resource/xml is loaded if it exists and **DDS_ProfileQos↩ Policy::ignore_resource_profile** (p. 1259) in **DDS_ProfileQosPolicy** (p. 1257) is set to **DDS_BOOLEAN_↩ FALSE** (p. 513) (first to be loaded). An example file, `NDDS_QOS_PROFILES.example.xml`, is available for reference.

- The URL groups separated by semicolons referenced by the enviroment variable NDDS_QOS_PROFILES are loaded if they exist and **DDS_ProfileQosPolicy::ignore_environment_profile** (p. 1258) in **DDS_ProfileQos↩ Policy** (p. 1257) is set to **DDS_BOOLEAN_FALSE** (p. 513).

- The file USER_QOS_PROFILES.xml in the working directory will be loaded if it exists and **DDS_ProfileQos↩ Policy::ignore_user_profile** (p. 1258) in **DDS_ProfileQosPolicy** (p. 1257) is set to **DDS_BOOLEAN_FALSE** (p. 513).

- The URL groups referenced by **DDS_ProfileQosPolicy::url_profile** (p. 1258) in **DDS_ProfileQosPolicy** (p. 1257) will be loaded if specified.

- The sequence of XML strings referenced by **DDS_ProfileQosPolicy::string_profile** (p. 1258) will be loaded if specified (last to be loaded).

The above methods can be combined together.

7.87.2 URL

The location of the XML resources (only files and strings are supported) is specified using a URL (Uniform Resource Locator) format. For example:

File Specification: `file:///usr/local/default_dds.xml`

String Specification: str://"<dds><qos_library> . . . lt;/qos_library>&lt/dds>"

If the URL schema name is omitted, RTI Connext will assume a file name. For example:

File Specification: /usr/local/default_dds.xml

7.87.2.1 URL groups

To provide redundancy and fault tolerance, you can specify multiple locations for a single XML document via URL groups. The syntax of a URL group is as follows:

[URL1 | URL2 | URL2 | . . .| URLn]

For example:

[`file:///usr/local/default_dds.xml` | `file:///usr/local/alternative_default_dds.xml`]

Only one of the elements in the group will be loaded by RTI Connext, starting from the left.

Brackets are not required for groups with a single URL.

7.87.2.2 NDDS_QOS_PROFILES environment variable

The environment variable NDDS_QOS_PROFILES contains a list of URL groups separated by ';'

The URL groups referenced by the environment variable are loaded if they exist and **DDS_ProfileQosPolicy::ignore_environment_profile** (p. 1258) is set to **DDS_BOOLEAN_FALSE** (p. 513)

7.87.2.3 Built-In QoS Profiles

There are also a number of built-in QoS profiles that can be used without having to load any configurations from outside XML resources. For more information on these built-in profiles see **Built-in Qos Profiles** (p. 679).

For more information on XML Configuration, refer to the `User's Manual`.

7.88 Publication Example

A data publication example.

A data publication example.

7.88.1 A typical publication example

Prep

- Create user data types using rtiddsgen. See the `Code Generator User's Manual`.

Set up

- **Get the factory** (p. 395)
- **Set up participant** (p. 395)
- **Set up publisher** (p. 399)
- **Register user data type(s)** (p. 396)
- **Set up topic(s)** (p. 396)
- **Set up data writer(s)** (p. 400)

Adjust the desired quality of service (QoS)

- **Adjust QoS on entities as necessary** (p. 406)

Send data

- **Send data** (p. 401)

Tear down

- **Tear down data writer(s)** (p. 401)
- **Tear down topic(s)** (p. 397)
- **Tear down publisher** (p. 399)
- **Tear down participant** (p. 396)

7.89 Subscription Example

A data subscription example.

A data subscription example.

7.89.1 A typical subscription example

Prep

- Create user data types using rtiddsgen. See the `Code Generator User's Manual`.

Set up

- **Get the factory** (p. 395)
- **Set up participant** (p. 395)
- **Set up subscriber** (p. 401)
- **Register user data type(s)** (p. 396)
- **Set up topic(s)** (p. 396)
- **Set up data reader(s)** (p. 403)
- **Set up data reader** (p. 404) **OR Set up subscriber** (p. 402) to receive data

Adjust the desired quality of service (QoS)

- **Adjust QoS on entities as necessary** (p. 406)

Receive data

- Access received data either **via a reader** (p. 404) **OR via a subscriber** (p. 402) (possibly in a **ordered or coherent** (p. 403) manner)

Tear down

- **Tear down data reader(s)** (p. 406)
- **Tear down topic(s)** (p. 397)
- **Tear down subscriber** (p. 403)
- **Tear down participant** (p. 396)

7.90 Participant Use Cases

Working with domain participants.

Working with domain participants.

7.90.1 Turning off auto-enable of newly created participant(s)

- **Get the factory** (p. 395)

- Change the value of the **ENTITY_FACTORY** (p. 600) for the **DDSDomainParticipantFactory** (p. 1694)

```
DDS_DomainParticipantFactoryQos factory_qos;

if (factory->get_qos(factory_qos) != DDS_RETCODE_OK) {
     printf("***Error: failed to get domain participant factory qos\n");
}
/* Change the QosPolicy to create disabled participants */
factory_qos.entity_factory.autoenable_created_entities = DDS_BOOLEAN_FALSE;
if (factory->set_qos(factory_qos) != DDS_RETCODE_OK) {
    printf("***Error: failed to set domain participant factory qos\n");
}
```

7.90.2 Getting the factory

- Get the **DDSDomainParticipantFactory** (p. 1694) instance:

```
DDSDomainParticipantFactory* factory = NULL;
factory = DDSDomainParticipantFactory::get_instance();
if (factory == NULL) {
   // ... error
}
```

7.90.3 Setting up a participant

- **Get the factory** (p. 395)

- Create **DDSDomainParticipant** (p. 1619):

```
 DDS_DomainParticipantQos participant_qos;
    DDSDomainParticipantListener* participant_listener = NULL;
    DDS_ReturnCode_t retcode;
 // Set the initial peers. This list includes all the computers the
 // application may communicate with, along with the maximum number of
 // RTI Data Distribution Service participants that can concurrently
 // run on that computer. This list only needs to be a superset of the
 // actual list of computers and participants that will be running at
 // any time.
 const char* NDDS_DISCOVERY_INITIAL_PEERS[] = {
  "host1",
  "10.10.30.192",
  "1@localhost",
  "2@host2",
  "my://", /* all unicast addrs on transport plugins with alias "my" */
  "2@shmem://", /* shared memory */
  "FF00:ABCD::0",
  "sf://0/0/R", /* StarFabric transport plugin */
  "1@FF00:0:1234::0",
  "225.1.2.3",
  "3@225.1.0.55",
  "FAA0::0#0/0/R",
};
const DDS_Long NDDS_DISCOVERY_INITIAL_PEERS_LENGTH =
      sizeof(NDDS_DISCOVERY_INITIAL_PEERS)/sizeof(const char*);
// initialize participant_qos with default values
retcode = factory->get_default_participant_qos(participant_qos);
if (retcode != DDS_RETCODE_OK) {
    printf("***Error: failed to get default participant qos\n");
}
if (!participant_qos.discovery.initial_peers.from_array(
                        NDDS_DISCOVERY_INITIAL_PEERS,
                         NDDS_DISCOVERY_INITIAL_PEERS_LENGTH)) {
    printf("***Error: failed to set discovery.initial_peers qos\n");
}
// Create the participant
```

```
DDSDomainParticipant* participant =
    factory->create_participant(domain_id,
                                participant_qos,
                                participant_listener,
                                        DDS_STATUS_MASK_NONE);
if (participant == NULL) {
     printf("***Error: failed to create domain participant\n");
};
return participant;
```

7.90.4 Tearing down a participant

- **Get the factory** (p. 395)
- Delete **DDSDomainParticipant** (p. 1619):
```
DDS_ReturnCode_t retcode;
retcode = factory->delete_participant(participant);
if (retcode != DDS_RETCODE_OK) {
    // ... check for cause of failure
}
```

7.91 Topic Use Cases

Working with topics.

Working with topics.

7.91.1 Registering a user data type

- **Set up participant** (p. 395)
- Register user data type of type `T` under the name "My_Type"
```
const char* type_name = "My_Type";
DDS_ReturnCode_t retcode;
retcode = FooTypeSupport::register_type(participant, type_name);
if (retcode != DDS_RETCODE_OK) {
    // ... error
}
```

7.91.2 Setting up a topic

- **Set up participant** (p. 395)
- **Ensure user data type is registered** (p. 396)
- Create a **DDSTopic** (p. 1885) under the name "my_topic"
```
const char* topic_name = "my_topic";
const char* type_type = "My_Type";   // user data type
DDS_TopicQos topic_qos;
DDS_ReturnCode_t retcode;
// MyTopicListener is user defined and
// extends DDSTopicListener
DDSTopicListener* topic_listener = new MyTopicListener(); // or = NULL
retcode = participant->get_default_topic_qos(topic_qos);
if (retcode != DDS_RETCODE_OK) {
    // ... error
}
DDSTopic* topic = participant->create_topic(topic_name, type_name,
                                    topic_qos, topic_listener,
                                    DDS_STATUS_MASK_ALL);
if (topic == NULL) {
    // ... error
};
```

7.91.3 Tearing down a topic

- Delete **DDSTopic** (p. 1885):
```
DDS_ReturnCode_t retcode;
retcode = participant->delete_topic(topic);
if (retcode != DDS_RETCODE_OK) {
    // ... check for cause of failure
}
```

7.92 FlowController Use Cases

Working with flow controllers.

Working with flow controllers.

7.92.1 Creating a flow controller

- **Set up participant** (p. 395)
- Create a flow controller
```
DDS_ReturnCode_t retcode;
DDSFlowController *controller = NULL;
DDS_FlowControllerProperty_t property;
retcode = participant->get_default_flowcontroller_property(property);
if (retcode != DDS_RETCODE_OK) {
    printf("***Error: failed to get default flow controller property\n");
}
// optionally modify flow controller property values
controller = participant->create_flowcontroller(
    "my flow controller name", property);
if (controller == NULL) {
    printf("***Error: failed to create flow controller\n");
}
```

7.92.2 Flow controlling a data writer

- **Set up participant** (p. 395)
- **Create flow controller** (p. 397)
- Create an asynchronous data writer, **FooDataWriter** (p. 1944), of user data type **Foo** (p. 1916):
```
DDS_DataWriterQos writer_qos;
DDS_ReturnCode_t retcode;
// MyWriterListener is user defined and
// extends DDSDataWriterListener
MyWriterListener* writer_listener = new MyWriterListener(); // or = NULL
retcode = publisher->get_default_datawriter_qos(writer_qos);
if (retcode != DDS_RETCODE_OK) {
    // ... error
}
/* Change the writer QoS to publish asnychronously */
writer_qos.publish_mode.kind = DDS_ASYNCHRONOUS_PUBLISH_MODE_QOS;
/* Setup to use the previously created flow controller */
writer_qos.publish_mode.flow_controller_name =
    DDS_String_dup("my flow controller name");
/* Samples queued for asynchronous write are subject to the History Qos policy */
writer_qos.history.kind = DDS_KEEP_ALL_HISTORY_QOS;
FooDataWriter* writer = publisher->create_datawriter(topic,
                                        writer_qos,
                                        writer_listener,
                        DDS_STATUS_MASK_ALL);
```

```
if (writer == NULL) {
    // ... error
};
/* Send data asynchronously... */
/* Wait for asynchronous send completes, if desired */
retcode = writer->wait_for_asynchronous_publishing(timout);
if (retcode != DDS_RETCODE_OK) {
    printf("***Error: failed to wait for asynchronous publishing\n");
}
```

7.92.3 Using the built-in flow controllers

RTI Connext provides several built-in flow controllers.

The **DDS_DEFAULT_FLOW_CONTROLLER_NAME** (p. 125) built-in flow controller provides the basic asynchronous writer behavior. When calling **FooDataWriter::write** (p. 1951), the call signals the **DDSPublisher** (p. 1818) asynchronous publishing thread (**DDS_PublisherQos::asynchronous_publisher** (p. 1279)) to send the actual data. As with any **DDS_ASYNCHRONOUS_PUBLISH_MODE_QOS** (p. 629) **DDSDataWriter** (p. 1589), the **FooDataWriter::write** (p. 1951) call returns immediately afterwards. The data is sent immediately in the context of the **DDSPublisher** (p. 1818) asynchronous publishing thread.

When using the **DDS_FIXED_RATE_FLOW_CONTROLLER_NAME** (p. 126) flow controller, data is also sent in the context of the **DDSPublisher** (p. 1818) asynchronous publishing thread, but at a regular fixed interval. The thread accumulates samples from different **DDSDataWriter** (p. 1589) instances and generates data on the wire only once per **DDS_FlowControllerTokenBucketProperty_t::period** (p. 1159).

In contrast, the **DDS_ON_DEMAND_FLOW_CONTROLLER_NAME** (p. 127) flow controller permits flow only when **DDSFlowController::trigger_flow** (p. 1738) is called. The data is still sent in the context of the **DDSPublisher** (p. 1818) asynchronous publishing thread. The thread accumulates samples from different **DDSDataWriter** (p. 1589) instances (across any **DDSPublisher** (p. 1818)) and sends all data since the previous trigger.

The properties of the built-in **DDSFlowController** (p. 1736) instances can be adjusted.

- **Set up participant** (p. 395)
- Lookup built-in flow controller

```
DDSFlowController *controller = NULL;
controller = participant->lookup_flowcontroller(
    DDS_DEFAULT_FLOW_CONTROLLER_NAME);
/* This should never happen, built-in flow controllers are always created */
if (controller == NULL) {
    printf("***Error: failed to lookup flow controller\n");
}
```

- Change property of built-in flow controller, if desired

```
DDS_ReturnCode_t retcode;
DDS_FlowControllerProperty_t property;
/* Get the property of the flow controller */
retcode = controller->get_property(property);
if (retcode != DDS_RETCODE_OK) {
    printf("***Error: failed to get flow controller property\n");
}
/* Change the property value as desired */
property.token_bucket.period.sec = 2;
property.token_bucket.period.nanosec = 0;
/* Update the flow controller property */
retcode = controller->set_property(property);
if (retcode != DDS_RETCODE_OK) {
    printf("***Error: failed to set flow controller property\n");
}
```

- **Create a data writer using the correct flow controller name** (p. 397)

7.92.4 Shaping the network traffic for a particular transport

- **Set up participant** (p. 395)
- **Create the transports** (p. 409)
- **Create a separate flow controller for each transport** (p. 397)
- Configure **DDSDataWriter** (p. 1589) instances to only use a single transport
- **Associate all data writers using the same transport to the corresponding flow controller** (p. 397)
- For each transport, the corresponding flow controller limits the network traffic based on the token bucket properties

7.92.5 Coalescing multiple samples in a single network packet

- **Set up participant** (p. 395)
- **Create a flow controller with a desired token bucket period** (p. 397)
- **Associate the data writer with the flow controller** (p. 397)
- Multiple samples written within the specified period will be coalesced into a single network packet (provided that `tokens_added_per_period` and `bytes_per_token` permit).

7.93 Publisher Use Cases

Working with publishers.

Working with publishers.

7.93.1 Setting up a publisher

- **Set up participant** (p. 395)
- Create a **DDSPublisher** (p. 1818)

```
DDS_PublisherQos publisher_qos;
// MyPublisherListener is user defined and
// extends DDSPublisherListener
DDSPublisherListener* publisher_listener =
        =                    new MyPublisherListener(); // or = NULL
participant->get_default_publisher_qos(publisher_qos);
DDSPublisher* publisher = participant->create_publisher(publisher_qos,
                                                        publisher_listener,
                                                       DDS_STATUS_MASK_ALL);
if (publisher == NULL) {
    // ... error
};
```

7.93.2 Tearing down a publisher

- Delete **DDSPublisher** (p. 1818):

```
DDS_ReturnCode_t retcode;
retcode = participant->delete_publisher(publisher);
if (retcode != DDS_RETCODE_OK) {
    // ... check for cause of failure
}
```

7.94 DataWriter Use Cases

Working with data writers.

Working with data writers.

7.94.1 Setting up a data writer

- **Set up publisher** (p. 399)
- **Set up a topic** (p. 396)
- Create a data writer, **FooDataWriter** (p. 1944), of user data type **Foo** (p. 1916) **:**

```
DDS_DataWriterQos writer_qos;
DDS_ReturnCode_t retcode;
// MyWriterListener is user defined and
// extends DDSDataWriterListener
MyWriterListener* writer_listener = new MyWriterListener(); // or = NULL
retcode = publisher->get_default_datawriter_qos(writer_qos);
if (retcode != DDS_RETCODE_OK) {
    // ... error
}
FooDataWriter* writer = publisher->create_datawriter(topic,
                                          writer_qos,
                                          writer_listener,
                                            DDS_STATUS_MASK_ALL);
if (writer == NULL) {
    // ... error
};
```

7.94.2 Managing instances

- Getting an instance "key" value of user data type **Foo** (p. 1916)

```
Foo* data = ...;     // user data
retcode = writer->get_key_value(*data, instance_handle);
if (retcode != DDS_RETCODE_OK) {
    // ... check for cause of failure
}
```

- Registering an instance of type **Foo** (p. 1916)

```
DDS_InstanceHandle_t instance_handle = DDS_HANDLE_NIL;
instance_handle = writer->register_instance(data);
```

- Unregistering an instance of type **Foo** (p. 1916)

```
retcode = writer->unregister_instance(data, instance_handle);
if (retcode != DDS_RETCODE_OK) {
    // ... check for cause of failure
}
```

- Disposing of an instance of type **Foo** (p. 1916)

```
retcode = writer->dispose(data, instance_handle);
if (retcode != DDS_RETCODE_OK) {
    // ... check for cause of failure
}
```

7.94.3 Sending data

- **Set up data writer** (p. 400)
- **Register instance** (p. 400)
- Write instance of type **Foo** (p. 1916)

```
Foo* data;     // user data
DDS_InstanceHandle_t instance_handle =
                 DDS_HANDLE_NIL; // or a valid registered handle
DDS_ReturnCode_t retcode;
retcode = writer->write(data, instance_handle);
if (retcode != DDS_RETCODE_OK) {
    // ... check for cause of failure
}
```

7.94.4 Tearing down a data writer

- Delete **DDSDataWriter** (p. 1589):

```
DDS_ReturnCode_t retcode;
retcode = publisher->delete_datawriter(writer);
if (retcode != DDS_RETCODE_OK) {
    // ... check for cause of failure
}
```

7.95 Subscriber Use Cases

Working with subscribers.

Working with subscribers.

7.95.1 Setting up a subscriber

- **Set up participant** (p. 395)
- Create a **DDSSubscriber** (p. 1860)

```
DDS_SubscriberQos subscriber_qos;
DDS_ReturnCode_t retcode;
// MySubscriberListener is user defined and
// extends DDSSubscriberListener
DDSSubscriberListener* subscriber_listener =
                               new MySubscriberListener(); // or = NULL
retcode = participant->get_default_subscriber_qos(subscriber_qos);
if (retcode != DDS_RETCODE_OK) {
    // ... error
};
DDSSubscriber* subscriber =
                participant->create_subscriber(subscriber_qos,
                                               subscriber_listener,
                                               DDS_STATUS_MASK_ALL);
if (subscriber == NULL) {
    // ... error
};
```

7.95.2 Set up subscriber to access received data

- **Set up subscriber** (p. 401)
- Set up to handle the DDS_DATA_ON_READERS_STATUS status, in one or both of the following two ways.
- **Enable DDS_DATA_ON_READERS_STATUS for the DDSSubscriberListener associated with the subscriber** (p. 407)
  - The processing to handle the status change is done in the **DDSSubscriberListener::on_data_on_readers()** (p. 1882) method of the attached listener.
  - Typical processing will **access the received data** (p. 402), either in arbitrary order or in a **coherent and ordered manner** (p. 403).
- **Enable DDS_DATA_ON_READERS_STATUS for the DDSStatusCondition associated with the subscriber** (p. 407)
  - The processing to handle the status change is done **when the subscriber's attached status condition is triggered** (p. 409) and the DDS_DATA_ON_READERS_STATUS status on the subscriber is changed.
  - Typical processing will **access the received data** (p. 402), either in an arbitrary order or in a **coherent and ordered manner** (p. 403).

7.95.3 Access received data via a subscriber

- **Ensure subscriber is set up to access received data** (p. 402)
- Get the list of readers that have data samples available:

```
DDSDataReaderSeq reader_seq;  // holder for list/set of readers
DDS_SampleStateMask     sample_state_mask = DDS_NOT_READ_SAMPLE_STATE;
DDS_ViewStateMask       view_state_mask = DDS_ANY_VIEW_STATE;
DDS_InstanceStateMask   instance_state_mask = DDS_ANY_INSTANCE_STATE;
DDS_ReturnCode_t retcode;
// get_datareadersX is not supported yet.
retcode = subscriber->get_datareaders(reader_seq,
                                      sample_state_mask,
                                      view_state_mask,
                                      instance_state_mask);
if (retcode != DDS_RETCODE_OK) {
    // ... check for cause of failure
}
```

- Upon successfully getting the list of readers with data, process the data readers to either:
  - **Read the data in each reader** (p. 405), **OR**
  - **Take the data in each reader** (p. 404)

  If the intent is to access the data **coherently or in order** (p. 403), the list of data readers *must* be processed in the order returned:

```
for(int i = 0; i < reader_seq.length(); ++i) {

  TDataReader* reader = reader_seq[i];
  // Take the data from reader,
  //              OR
  // Read the data from reader
}
```

- **Alternatively**, call **DDSSubscriber::notify_datareaders** (p. 1876) to invoke the **DDSDataReaderListener** (p. 1583) for each of the data readers.

```
retcode = subscriber->notify_datareaders();
if (retcode != DDS_RETCODE_OK) {
    // ... check for cause of failure
}
```

7.95.4 Access received data coherently and/or in order

To access the received data coherently and/or in an ordered manner, according to the settings of the **DDS_↩ PresentationQosPolicy** (p. 1249) attached to a **DDSSubscriber** (p. 1860):

- **Ensure subscriber is set up to access received data** (p. 402)
- Indicate that data will be accessed via the subscriber:
```
retcode = subscriber->begin_access();
if (retcode != DDS_RETCODE_OK) {
    // ... check for cause of failure
}
```
- **Access received data via the subscriber, making sure that the data readers are processed in the order returned.** (p. 402)
- Indicate that the data access via the subscriber is done:
```
retcode = subscriber->end_access();
if (retcode != DDS_RETCODE_OK) {
    // ... check for cause of failure
}
```

7.95.5 Tearing down a subscriber

- Delete **DDSSubscriber** (p. 1860):
```
DDS_ReturnCode_t retcode;
retcode = participant->delete_subscriber(subscriber);
if (retcode != DDS_RETCODE_OK) {
    // ... check for cause of failure
}
```

7.96 DataReader Use Cases

Working with data readers.

Working with data readers.

7.96.1 Setting up a data reader

- **Set up subscriber** (p. 401)
- **Set up a topic** (p. 396)
- Create a data reader, **FooDataReader** (p. 1917), of user data type **Foo** (p. 1916):
```
DDS_DataReaderQos reader_qos;
DDS_ReturnCode_t retcode;
// MyReaderListener is user defined and
// extends DDSDataReaderListener
DDSDataReaderListener* reader_listener =
      new MyReaderListener(); // or = NULL
retcode = subscriber->get_default_datareader_qos(reader_qos);
if (retcode != DDS_RETCODE_OK) {
    // ... error
};
FooDataReader* reader = subscriber->create_datareader(topic,
                                                      reader_qos,
                                                      reader_listener,
                                                  DDS_STATUS_MASK_ALL);
if (reader == NULL) {
    // ... error
};
```

7.96.2 Managing instances

- Given a data reader
```
FooDataReader* reader = ...;
```
- Getting an instance "key" value of user data type **Foo** (p. 1916)
```
Foo data;     // user data of type Foo
// ...
retcode = reader->get_key_value(data, instance_handle);
if (retcode != DDS_RETCODE_OK) {
    // ... check for cause of failure
}
```

7.96.3 Set up reader to access received data

- **Set up data reader** (p. 403)
- Set up to handle the DDS_DATA_AVAILABLE_STATUS status, in one or both of the following two ways.
- **Enable DDS_DATA_AVAILABLE_STATUS for the DDSDataReaderListener associated with the data reader** (p. 407)
  - The processing to handle the status change is done in the **DDSDataReaderListener::on_data_available** (p. 1585) method of the attached listener.
  - Typical processing will **access the received data** (p. 404).
- **Enable DDS_DATA_AVAILABLE_STATUS for the DDSStatusCondition associated with the data reader** (p. 407)
  - The processing to handle the status change is done **when the data reader's attached status condition is triggered** (p. 409) and the DDS_DATA_AVAILABLE_STATUS status on the data reader is changed.
  - Typical processing will **access the received data** (p. 404).

7.96.4 Access received data via a reader

- **Ensure reader is set up to access received data** (p. 404)
- Access the received data, by either:
  - **Taking the received data in the reader** (p. 404), **OR**
  - **Reading the received data in the reader** (p. 405)

7.96.5 Taking data

- **Ensure reader is set up to access received data** (p. 404)

- Take samples of user data type `T`. The samples are removed from the Service. The caller is responsible for deallocating the buffers.

```
FooSeq                data_seq; // holder for sequence of user data type Foo
DDS_SampleInfoSeq     info_seq; // holder for sequence of DDS_SampleInfo
long                  max_samples       = DDS_LENGTH_UNLIMITED;
DDS_SampleStateMask   sample_state_mask = DDS_ANY_SAMPLE_STATE;
DDS_ViewStateMask     view_state_mask = DDS_ANY_VIEW_STATE;
DDS_InstanceStateMask instance_state_mask = DDS_ANY_INSTANCE_STATE;
DDS_ReturnCode_t retcode;
retcode = reader->take(data_seq, info_seq,
                 max_samples,
                       sample_state_mask,
                       view_state_mask,
                       instance_state_mask);
if (retcode == DDS_RETCODE_NO_DATA) {
    return;
} else if (retcode != DDS_RETCODE_OK) {
    // ... check for cause of failure
}
```

- Use the received data

```
// Use the received data samples 'data_seq' and associated information 'info_seq'
for(int i = 0; i < data_seq.length(); ++i) {
    // use... data_seq[i] ...
    // use... info_seq[i] ...
}
```

- Return the data samples and the information buffers back to the middleware. *IMPORTANT*: Once this call returns, you must not retain any pointers to any part of any sample or sample info object.

```
retcode = reader->return_loan(data_seq, info_seq);
if (retcode != DDS_RETCODE_OK) {
    // ... check for cause of failure
}
```

7.96.6 Reading data

- **Ensure reader is set up to access received data** (p. 404)

- Read samples of user data type **`Foo`** (p. 1916). The samples are not removed from the Service. It remains responsible for deallocating the buffers.

```
FooSeq                data_seq; // holder for sequence of user data type Foo
DDS_SampleInfoSeq      info_seq; // holder for sequence of DDS_SampleInfo
long                  max_samples       = DDS_LENGTH_UNLIMITED;
DDS_SampleStateMask   sample_state_mask = DDS_NOT_READ_SAMPLE_STATE;
DDS_ViewStateMask     view_state_mask = DDS_ANY_VIEW_STATE;
DDS_InstanceStateMask instance_state_mask = DDS_ANY_INSTANCE_STATE;
DDS_ReturnCode_t retcode;
retcode = reader->read(data_seq, info_seq,
                 max_samples,
                       sample_state_mask,
                       view_state_mask,
                       instance_state_mask);
if (retcode == DDS_RETCODE_NO_DATA) {
    return;
} else if (retcode != DDS_RETCODE_OK) {
    // ... check for cause of failure
}
```

- Use the received data

```
// Use the received data samples 'data_seq' and associated
// information 'info_seq'
for(int i = 0; i < data_seq.length(); ++i) {
    // use... data_seq[i] ...
    // use... info_seq[i] ...
}
```

- Return the data samples and the information buffers back to the middleware

```
retcode = reader->return_loan(data_seq, info_seq);
if (retcode != DDS_RETCODE_OK) {
    // ... check for cause of failure
}
```

7.96.7 Tearing down a data reader

- Delete **DDSDataReader** (p. 1556):
```
DDS_ReturnCode_t retcode;
retcode = subscriber->delete_datareader(reader);
if (retcode != DDS_RETCODE_OK) {
    // ... check for cause of failure
}
```

7.97 Entity Use Cases

Working with entities.

Working with entities.

7.97.1 Enabling an entity

- To enable an **DDSEntity** (p. 1731)
```
    if (entity->enable() != DDS_RETCODE_OK) {
        printf("***Error: failed to enable entity\n");
    }
```

7.97.2 Checking if a status changed on an entity.

- Given an **DDSEntity** (p. 1731) and a **DDS_StatusKind** (p. 538) to check for, get the list of statuses that have changed since the last time they were respectively cleared.
```
    DDS_StatusMask status_changes_mask = entity->get_status_changes();
```
- Check if `status_kind` was changed since the last time it was cleared. A plain communication status change is cleared when the status is read using the entity's `get_<plain communication status>()` method. A read communication status change is cleared when the data is taken from the middleware via a TDataReader_↩ take() call [see **Changes in Status** (p. 535) for details].
```
    if (status_changes_mask & status_kind) {
        return true;
    } else {           /* ... YES, status_kind changed ... */
        return false;  /* ... NO, status_kind did NOT change ... */
    }
```

7.97.3 Changing the QoS for an entity

The QoS for an entity can be specified at the entity creation time. Once an entity has been created, its QoS can be manipulated as follows.

- Get an entity's QoS settings using **get_qos (abstract)** (p. 1733)
```
    if (entity->get_qos(qos) != DDS_RETCODE_OK) {
        printf("***Error: failed to get qos\n");
    }
```
- Change the desired qos policy fields
```
    /* Change the desired qos policies */
    /* qos.policy.field = ... */
```

- Set the qos using **set_qos (abstract)** (p. 1732).
```
switch (entity->set_qos(qos)) {
    case DDS_RETCODE_OK: { /* success */
    } break;
    case DDS_RETCODE_IMMUTABLE_POLICY: {
        printf("***Error: tried changing a policy that can only be"
               "          set at entity creation time\n");
    } break;
    case DDS_RETCODE_INCONSISTENT_POLICY: {
        printf("***Error: tried changing a policy to a value inconsistent"
               "          with other policy settings\n");
    } break;
    default: {
        printf("***Error: some other failure\n");
    }
}
```

7.97.4 Changing the listener and enabling/disabling statuses associated with it

The listener for an entity can be specified at the entity creation time. By default the listener is *enabled* for all the statuses supported by the entity.

Once an entity has been created, its listener and/or the statuses for which it is enabled can be manipulated as follows.

- User defines entity listener methods
```
/* ... methods defined by EntityListener ... */
public class MyEntityListener implements DDSListener {
    // ... methods defined by EntityListener ...
}
```

- Get an entity's listener using **get_listener (abstract)** (p. 1734)
```
entity_listener = entity->get_listener();
```
- Enable `status_kind` for the `listener`
```
enabled_status_list |= status_kind;
```
- Disable `status_kind` for the `listener`
```
enabled_status_list &= ~status_kind;
```
- Set an entity's listener to `entity_listener` using **set_listener (abstract)** (p. 1733). Only enable the listener for the statuses specified by the `enabled_status_list`.
```
if (entity->set_listener(entity_listener, enabled_status_list)
            != DDS_RETCODE_OK) {
    printf("***Error: setting entity listener\n");
}
```

7.97.5 Enabling/Disabling statuses associated with a status condition

Upon entity creation, by default, all the statuses are *enabled* for the DDS_StatusCondition associated with the entity.

Once an entity has been created, the list of statuses for which the DDS_StatusCondition is triggered can be manipulated as follows.

- Given an `entity`, a `status_kind`, and the associated `status_condition`:
```
statuscondition = entity->get_statuscondition();
```
  - Get the list of statuses enabled for the `status_condition`
```
enabled_status_list = statuscondition->get_enabled_statuses();
```

- Check if the given `status_kind` is enabled for the `status_condition`

```
if (enabled_status_list & status_kind) {
    /*... YES, status_kind is enabled ... */
} else {
    /* ... NO, status_kind is NOT enabled ... */
}
```

- Enable `status_kind` for the `status_condition`

```
if (statuscondition->set_enabled_statuses(enabled_status_list | status_kind)
            != DDS_RETCODE_OK) {
    /* ... check for cause of failure */
}
```

- Disable `status_kind` for the `status_condition`

```
if (statuscondition->set_enabled_statuses(enabled_status_list & ~status_kind)
            != DDS_RETCODE_OK) {
    /* ... check for cause of failure */
}
```

7.98 Waitset Use Cases

Using wait-sets and conditions.

Using wait-sets and conditions.

7.98.1 Setting up a wait-set

- Create a wait-set

```
DDSWaitSet* waitset = new DDSWaitSet();
```

- Attach conditions

```
DDSCondition* cond1 = ...;
DDSCondition* cond2 = entity->get_statuscondition();
DDSCondition* cond3 = reader->create_readcondition(DDS_NOT_READ_SAMPLE_STATE,
                                                   DDS_ANY_VIEW_STATE,
                                                   DDS_ANY_INSTANCE_STATE);
DDSCondition* cond4 = new DDSGuardCondition();
DDSCondition* cond5 = ...;
DDS_ReturnCode_t retcode;
retcode = waitset->attach_condition(cond1);
if (retcode != DDS_RETCODE_OK) {
    // ... error
}
retcode = waitset->attach_condition(cond2);
if (retcode != DDS_RETCODE_OK) {
    // ... error
}
retcode = waitset->attach_condition(cond3);
if (retcode != DDS_RETCODE_OK) {
    // ... error
}
retcode = waitset->attach_condition(cond4);
if (retcode != DDS_RETCODE_OK) {
    // ... error
}
retcode = waitset->attach_condition(cond5);
if (retcode != DDS_RETCODE_OK) {
    // ... error
}
```

7.98.2 Waiting for condition(s) to trigger

- **Set up a wait-set** (p. 408)
- Wait for a condition to trigger or `timeout`, whichever occurs first

```
DDS_Duration_t timeout = { 0, 1000000 }; // 1ms
DDSConditionSeq active_conditions; // holder for active conditions
bool is_cond1_triggered = false;
bool is_cond2_triggered = false;
DDS_ReturnCode_t retcode;
retcode = waitset->wait(active_conditions, timeout);
if (retcode != DDS_RETCODE_OK) {
    if (retcode == DDS_RETCODE_TIMEOUT) {
        // Timeout
    } else {
        // Failure
} else {
    // success, check if "cond1" or "cond2" are triggered:
        for(int i = 0; i < active_conditions.length(); ++i) {
           if (active_conditions[i] == cond1) {
              printf("Cond1 was triggered!");
              is_cond1_triggered = true;
           }
           if (active_conditions[i] == cond2) {
              printf("Cond2 was triggered!");
              is_cond2_triggered = true;
           }
           if (is_cond1_triggered && is_cond2_triggered) {
              break;
           }
       }
}
if (is_cond1_triggered) {
    // ... do something because "cond1" was triggered ...
}
if (is_cond2_triggered) {
    // ... do something because "cond2" was triggered ...
}
return retcode;
```

7.98.3 Tearing down a wait-set

- Delete the wait-set

```
delete waitset;
waitset = NULL;
```

7.99 Transport Use Cases

Working with pluggable transports.

Working with pluggable transports.

7.99.1 Changing the automatically registered built-in transports

- The **DDS_TRANSPORTBUILTIN_MASK_DEFAULT** (p. 642) specifies the transport plugins that will be automatically registered with a newly created **DDSDomainParticipant** (p. 1619) by default.
- This default can be changed by changing the value of the value of **TRANSPORT_BUILTIN** (p. 641) Qos Policy on the **DDSDomainParticipant** (p. 1619)
- To change the **DDS_DomainParticipantQos::transport_builtin** (p. 974) Qos Policy:

```
    DDS_DomainParticipantQos participant_qos;

    factory->get_default_participant_qos(participant_qos);

    participant_qos.transport_builtin.mask = DDS_TRANSPORTBUILTIN_SHMEM |
                                             DDS_TRANSPORTBUILTIN_UDPv4;
```

7.99.2 Changing the properties of the automatically registered builtin transports

The behavior of the automatically registered builtin transports can be altered by changing their properties.

- Tell the **DDSDomainParticipantFactory** (p. 1694) to create the participants disabled, as described in **Turning off auto-enable of newly created participant(s)** (p. 395)
- Get the property of the desired builtin transport plugin, say ::**UDPv4 Transport** (p. 462)

```
struct NDDS_Transport_UDPv4_Property_t property = NDDS_TRANSPORT_UDPV4_PROPERTY_DEFAULT;

if (NDDSTransportSupport::get_builtin_transport_property(
                                    participant,
                                    DDS_TRANSPORTBUILTIN_UDPv4,
                                    (struct NDDS_Transport_Property_t&)property)
       != DDS_RETCODE_OK) {
    printf("***Error: get builtin transport property\n");
}
```

- Change the property fields as desired. Note that the properties should be changed carefully, as inappropriate values may prevent communications. For example, the ::**UDPv4 Transport** (p. 462) properties can be changed to support **large messages** (assuming the underlying operating system's UDPv4 stack supports the large message size). Note: if message_size_max is increased from the default for any of the built-in transports, then the **DDS_ReceiverPoolQosPolicy::buffer_size** (p. 1294) on the DomainParticipant should also be changed.

```
/* Decrease the UDPv4 maximum message size to 9K (small messages). */
property.parent.message_size_max =  9216;
property.recv_socket_buffer_size =  9216;
property.send_socket_buffer_size =  9216;
```

- Set the property of the desired builtin transport plugin, say ::**UDPv4 Transport** (p. 462)

```
if (NDDSTransportSupport::set_builtin_transport_property(
                                    participant,
                                    DDS_TRANSPORTBUILTIN_UDPv4,
                                    (struct NDDS_Transport_Property_t&)property)
       != DDS_RETCODE_OK) {
    printf("***Error: set builtin transport property\n");
}
```

- **Enable the participant** (p. 406) to turn on communications with other participants in the domain using the new properties for the automatically registered builtin transport plugins.

7.99.3 Creating a transport

- A transport plugin is created using methods provided by the supplier of the transport plugin.
- For example to create an instance of the ::**UDPv4 Transport** (p. 462)

```
NDDS_Transport_Plugin* transport = NULL;
struct NDDS_Transport_UDPv4_Property_t property = NDDS_TRANSPORT_UDPV4_PROPERTY_DEFAULT;
transport = NDDS_Transport_UDPv4_new(&property);
if (transport == NULL) {
   printf("***Error: creating transport plugin\n");
}
```

7.99.4 Deleting a transport

- A transport plugin can only be deleted only after the **DDSDomainParticipant** (p. 1619) with which it is registered is deleted.
- The virtual destructor provided by the abstract transport plugin API can be used to delete a transport plugin.

```
transport->delete_cEA(transport, NULL);
```

7.99.5 Registering a transport with a participant

The basic steps for setting up transport plugins for use in an RTI Connext application are described below.

- Tell the **DDSDomainParticipantFactory** (p. 1694) to create the participants disabled, as described in **Turning off auto-enable of newly created participant(s)** (p. 395)

Optionally **Changing the automatically registered built-in transports** (p. 409)

Optionally **Changing the properties of the automatically registered builtin transports** (p. 410)

- Create a disabled **DDSDomainParticipant** (p. 1619), as described in **Setting up a participant** (p. 395)
- Decide on the **network address** (p. 176) for the transport plugin. The network address should be chosen so that the resulting fully qualified address is globally unique (across all transports used in the domain).

```
/* Decide on a network address (96 bits for UDPv4), such that the fully
   qualified unicast address for the transport's interfaces will be
   globally unique. For example, we use the network address:
            1234:1234:1234:0000
   It will be prepended to the unicast addresses of the transport plugin's
   interfaces, to give a fully qualified address that is unique in the
   domain.
*/
NDDS_Transport_Address_t network_address = {{1,2,3,4, 1,2,3,4, 1,2,3,4, 0,0,0,0}};
```

- Decide on the **aliases** (p. 173) for the transport plugin. An alias can refer to one or more transport plugins. The transport class name (see **Builtin Transport Class Names** (p. 668)) are automatically appended to the user-provided aliases. Alias names are useful in creating logical groupings of transports, e.g. all the transports that are configured to support large messages may be given the alias "large_message".

```
/* Decide aliases, i.e. the names by which this transport plugin will be known */
const char* ALIASES[] = {
    "my",
    "large_message",
};
const DDS_Long ALIASES_LENGTH = sizeof(ALIASES)/sizeof(const char*);

/* Initialize the aliases StringSeq */
DDS_StringSeq aliases;
if (!aliases.from_array(ALIASES, ALIASES_LENGTH)) {
    printf("***Error: creating initializing aliases\n");
}
```

- Register the transport plugin with the **DDSDomainParticipant** (p. 1619). Note that a transport plugin should **NOT be registered with more than one DomainParticipant**. It is the responsibility of the application programmer to ensure that this requirement is not violated.

```
NDDS_Transport_Handle_t handle = NDDS_TRANSPORT_HANDLE_NIL;

handle = NDDSTransportSupport::register_transport(
                        participant,          /* Disabled Domain Participant */
                        transport,            /* Transport plugin */
                        aliases,              /* Transport aliases */
                        network_address);     /* Transport network address */
if (NDDS_Transport_Handle_is_nil(&handle)) {
   printf("***Error: registering transport\n");
}
```

Optionally **Adding receive routes for a transport** (p. 412)

Optionally **Adding send routes for a transport** (p. 412)

- **Enable the participant** (p. 406) to turn on communications with other participants in the domain, using the newly registered transport plugins, and automatically registered builtin transport plugins (if any).

7.99.6 Adding receive routes for a transport

- Receive routes can be added to restrict address ranges on which incoming messages can be received. Any number of receive routes can be added, but these must be done before the participant is enabled.
- To restrict the address range from which incoming messages can be received by the transport plugin:

```
/* Restrict to receiving messages only on interfaces
           1234:1234:1234:10.10.*.*
*/
NDDS_Transport_Address_t subnet = {{1,2,3,4, 1,2,3,4, 1,2,3,4, 10,10,0,0}};
if (NDDSTransportSupport::add_receive_route(handle, subnet, 112)
                          != DDS_RETCODE_OK) {
   printf("***Error: adding receive route\n");
}
```

7.99.7 Adding send routes for a transport

- Send routes can be added to restrict the address ranges to which outgoing messages can be sent by the transport plugin. Any number of send routes can be added, but these must be done before the participant is enabled.
- To restrict address ranges to which outgoing messages can be sent by the transport plugin:

```
/* Restrict to sending messages only to addresses (subnets)
             1234:1234:1234:10.10.30.*
*/
NDDS_Transport_Address_t subnet = {{1,2,3,4, 1,2,3,4, 1,2,3,4, 10,10,30,0}};
if (NDDSTransportSupport::add_send_route(handle, subnet, 120)
                          != DDS_RETCODE_OK) {
   printf("***Error: adding send route\n");
}
```

7.100 Filter Use Cases

Working with data filters.

Working with data filters.

7.100.1 Introduction

RTI Connext supports filtering data either during the exchange from **DDSDataWriter** (p. 1589) to **DDSDataReader** (p. 1556), or after the data has been stored at the **DDSDataReader** (p. 1556).

Filtering during the exchange process is performed by a **DDSContentFilteredTopic** (p. 1551), which is created by the **DDSDataReader** (p. 1556) as a way of specifying a subset of the data samples that it wishes to receive.

Filtering samples that have already been received by the **DDSDataReader** (p. 1556) is performed by creating a **DDSQueryCondition** (p. 1841), which can then used to check for matching samples, be alerted when matching samples arrive, or retrieve matching samples through use of the **FooDataReader::read_w_condition** (p. 1925) or **FooData↩Reader::take_w_condition** (p. 1926) functions. (Conditions may also be used with the APIs **FooDataReader::read↩_next_instance_w_condition** (p. 1938) and **FooDataReader::take_next_instance_w_condition** (p. 1939).)

Filtering may be performed on any topic, either keyed or un-keyed, except the **Built-in Topics** (p. 61). Filtering may be perfomed on any field, subset of fields, or combination of fields, subject only to the limitations of the filter syntax, and some restrictions against filtering some *sparse value types* of the **Dynamic Data** (p. 103) API.

RTI Connext contains built in support for filtering using SQL syntax, described in the **Queries and Filters Syntax** (p. 184) module.

7.100.1.1 Overview of ContentFilteredTopic

Each **DDSContentFilteredTopic** (p. 1551) is created based on an existing **DDSTopic** (p. 1885). The **DDSTopic** (p. 1885) specifies the **field_names** and **field_types** of the data contained within the topic. The **DDSContentFiltered↩ Topic** (p. 1551), by means of its `filter_expression` and `expression_parameters`, further specifies the *values* of the data which the **DDSDataReader** (p. 1556) wishes to receive.

Custom filters may also be constructed and utilized as described in the **Creating Custom Content Filters** (p. 415) module.

Once the **DDSContentFilteredTopic** (p. 1551) has been created, a **DDSDataReader** (p. 1556) can be created using the filtered topic. The filter's characteristics are exchanged between the **DDSDataReader** (p. 1556) and any matching **DDSDataWriter** (p. 1589) during the discovery process.

If the **DDSDataWriter** (p. 1589) allows (by **DDS_DataWriterResourceLimitsQosPolicy::max_remote_reader_filters** (p. 920)), and the **DDSDataReader** (p. 1556) 's **DDS_TransportMulticastQosPolicy** (p. 1415) is empty, then the **DDSDataWriter** (p. 1589) will filter out samples that don't meet the filtering criteria.

If disallowed by the **DDSDataWriter** (p. 1589), or the **DDSDataReader** (p. 1556) has set the **DDS_TransportMulticast↩ QosPolicy** (p. 1415), then the **DDSDataWriter** (p. 1589) sends all samples to the **DDSDataReader** (p. 1556), and the **DDSDataReader** (p. 1556) discards any samples that do not meet the filtering criteria.

The `expression_parameters` can be modified using **DDSContentFilteredTopic::set_expression_parameters** (p. 1553). Any changes made to the filtering criteria by means of **DDSContentFilteredTopic::set_expression_↩ parameters** (p. 1553), will be conveyed to any connected **DDSDataWriter** (p. 1589). New samples will be subject to the modified filtering criteria, but samples that have already been accepted or rejected are unaffected. However, if the **DDSDataReader** (p. 1556) connects to a **DDSDataWriter** (p. 1589) that *re-sends* its data, the re-sent samples will be subjected to the new filtering criteria.

7.100.1.2 Overview of QueryCondition

DDSQueryCondition (p. 1841) combine aspects of the content filtering capabilities of **DDSContentFilteredTopic** (p. 1551) with state filtering capabilities of **DDSReadCondition** (p. 1842) to create a reconfigurable means of filtering or searching data in the **DDSDataReader** (p. 1556) queue.

DDSQueryCondition (p. 1841) may be created on a disabled **DDSDataReader** (p. 1556), or after the **DDSDataReader** (p. 1556) has been enabled. If the **DDSDataReader** (p. 1556) is enabled, and has already recevied and stored samples in its queue, then all data samples in the are filtered against the **DDSQueryCondition** (p. 1841) filter criteria at the time that the **DDSQueryCondition** (p. 1841) is created. (Note that an exclusive lock is held on the **DDSDataReader** (p. 1556) sample queue for the duration of the **DDSQueryCondition** (p. 1841) creation).

Once created, incoming samples are filtered against all **DDSQueryCondition** (p. 1841) filter criteria at the time of their arrival and storage into the **DDSDataReader** (p. 1556) queue.

The number of **DDSQueryCondition** (p. 1841) filters that an individual **DDSDataReader** (p. 1556) may create is set by **DDS_DataReaderResourceLimitsQosPolicy::max_query_condition_filters** (p. 880), to an upper maximum of 32.

7.100.2 Filtering with ContentFilteredTopic

- **Set up subscriber** (p. 401)
- **Set up a topic** (p. 396)
- Create a ContentFilteredTopic, of user data type **Foo** (p. 1916):

```
DDSContentFilteredTopic *cft = NULL;
DDS_StringSeq cft_parameters(2);
const char* cft_param_list[] = {"1", "100"};
cft_parameters.from_array(cft_param_list, 2);
cft = participant->create_contentfilteredtopic("ContentFilteredTopic",
                                               Foo_topic,
                                               "value > %0 AND value < %1",
                                               cft_parameters);
if (cft == NULL) {
    printf("create_contentfilteredtopic error\n");
    subscriber_shutdown(participant);
    return -1;
}
```

- Create a FooReader using the ContentFilteredTopic:

```
DDSDataReader *reader = NULL;
reader = subscriber->create_datareader(cft,
                                       datareader_qos, // or DDS_DATAREADER_QOS_DEFAULT
                                       reader_listener, // or NULL
                                       DDS_STATUS_MASK_ALL);
if (reader == NULL) {
    printf("create_datareader error\n");
    subscriber_shutdown(participant);
    return -1;
}
FooDataReader *Foo_reader = FooDataReader::narrow(reader);
if (Foo_reader == NULL) {
    printf("DataReader narrow error\n");
    subscriber_shutdown(participant);
    return -1;
}
```

Once setup, reading samples with a **DDSContentFilteredTopic** (p. 1551) is exactly the same as normal reads or takes, as decribed in **DataReader Use Cases** (p. 403).

- Changing filter crieria using set_expression_parameters:

```
cft->get_expression_parameters(cft_parameters);
DDS_String_free(cft_parameters[0]);
DDS_String_free(cft_parameters[1]);
cft_parameters[0] = DDS_String_dup("5");
cft_arameters[1] = DDS_String_dup("9");
retcode = cft->set_expression_parameters(cft_arameters);
if (retcode != DDS_RETCODE_OK) {
    printf("set_expression_parameters error\n");
    subscriber_shutdown(participant);
    return -1;
}
```

7.100.3 Filtering with Query Conditions

- Given a data reader of type **Foo** (p. 1916)

```
DDSDataReader *reader = ...;
FooDataReader *Foo_reader = FooDataReader::narrow(reader);
```

- Creating a QueryCondition

```
DDSQueryCondition *queryCondition = NULL;
DDS_StringSeq qc_parameters(2);
const char *qc_param_list[] = {"0","100"};
qc_parameters.from_array(qc_param_list,2);
queryCondition = reader->create_querycondition(DDS_NOT_READ_SAMPLE_STATE,
                                               DDS_ANY_VIEW_STATE,
                                               DDS_ALIVE_INSTANCE_STATE,
                                               "value > %0 AND value < %1",
                                               qc_parameters);
```

```
if (queryCondition == NULL) {
    printf("create_query_condition error\n");
    goto error_exit;
}
```

- Reading matching samples with a **DDSQueryCondition** (p. 1841)

```
FooSeq data_seq;
DDS_SampleInfoSeq info_seq;
retcode = Foo_reader->read_w_condition(data_seq, info_seq,
                                       DDS_LENGTH_UNLIMITED,
                                       queryCondition);
if (retcode == DDS_RETCODE_NO_DATA) {
    printf("no matching data\n");
} else if (retcode != DDS_RETCODE_OK) {
    printf("read_w_condition error %d\n", retcode);
    subscriber_shutdown(participant);
    return -1;
} else {
    for (i = 0; i < data_seq.length(); ++i) {
        if (info_seq[i].valid_data) {
            /* process your data here */
        }
    retcode = Foo_reader->return_loan(data_seq, info_seq);
    if (retcode != DDS_RETCODE_OK) {
        printf("return loan error %d\n", retcode);
        subscriber_shutdown(participant);
        return -1;
    }
}
```

- **DDSQueryCondition::set_query_parameters** (p. 1842) is used similarly to **DDSContentFilteredTopic::set_↩ expression_parameters** (p. 1553), and the same coding techniques can be used.
- Any **DDSQueryCondition** (p. 1841) that have been created must be deleted before the **DDSDataReader** (p. 1556) can be deleted. This can be done using **DDSDataReader::delete_contained_entities** (p. 1563) or manually as in:

```
retcode = reader->delete_readcondition(queryCondition);
```

7.100.4 Filtering Performance

Although RTI Connext supports filtering on any field or combination of fields using the SQL syntax of the built-in filter, filters for keyed topics that filter solely on the contents of key fields have the potential for much higher performance. This is because for key-only filters, the **DDSDataReader** (p. 1556) caches the results of the filter (pass or not pass) for each instance. When another sample of the same instance is seen at the **DDSDataReader** (p. 1556), the filter results are retrieved from the cache, dispensing with the need to call the filter function.

This optimization applies to all filtering using the built-in SQL filter, performed by the **DDSDataReader** (p. 1556), for either **DDSContentFilteredTopic** (p. 1551) or **DDSQueryCondition** (p. 1841). This does *not* apply to filtering perfomed for **DDSContentFilteredTopic** (p. 1551) by the **DDSDataWriter** (p. 1589).

7.101 Creating Custom Content Filters

Working with custom content filters.

Working with custom content filters.

7.101.1 Introduction

By default, RTI Connext creates content filters with the DDS_SQL_FILTER, which implements a superset of the DDS-specified SQL WHERE clause. However, in many cases this filter may not be what you want. Some examples are:

- The default filter can only filter based on the content of a sample, not on a computation on the content of a sample. You can use a custom filter that is customized for a specific type and can filter based on a computation of the type members.
- You want to use a different filter language then SQL

This HOWTO explains how to write your own custom filter and is divided into the following sections:

- **The Custom Content Filter API** (p. 416)
- **Example Using C format strings** (p. 417)

7.101.2 The Custom Content Filter API

A custom content filter is created by calling the **DDSDomainParticipant::register_contentfilter** (p. 1632) function with a **DDSContentFilter** (p. 1548) that contains a **compile**, an **evaluate** function and a **finalize** function. **DDSContentFilteredTopic** (p. 1551) can be created with **DDSDomainParticipant::create_contentfilteredtopic_↩ with_filter** (p. 1654) to use this filter.

A custom content filter is used by RTI Connext at the following times during the life-time of a **DDSContentFilteredTopic** (p. 1551) (the function called is shown in parenthesis).

- When a **DDSContentFilteredTopic** (p. 1551) is created (**compile** (p. 416))
- When the filter parameters are changed on the **DDSContentFilteredTopic** (p. 1551) (**compile** (p. 416)) with **DDSContentFilteredTopic::set_expression_parameters** (p. 1553)
- When a sample is filtered (**evaluate** (p. 416)). This function is called by the RTI Connext core with a de-serialized sample
- When a **DDSContentFilteredTopic** (p. 1551) is deleted (**finalize** (p. 417))

7.101.2.1 The compile function

The **compile** (p. 417) function is used to **compile** a filter expression and expression parameters. Please note that the term **compile** is intentionally loosely defined. It is up to the user to decide what this function should do and return.

See **DDSContentFilter::compile** (p. 1548) for details.

7.101.2.2 The evaluate function

The **evaluate** (p. 417) function is called each time a sample is received to determine if a sample should be filtered out and discarded.

See **DDSContentFilter::evaluate** (p. 1550) for details.

7.101.2.3 The finalize function

The **finalize** (p. 417) function is called when an instance of the custom content filter is no longer needed. When this function is called, it is safe to free all resources used by this particular instance of the custom content filter.

See **DDSContentFilter::finalize** (p. 1550) for details.

7.101.3 Example Using C format strings

Assume that you have a type **Foo** (p. 1916).

You want to write a custom filter function that will drop all samples where the value of Foo.x $>$ x and x is a value determined by an expression parameter. The filter will **only** be used to filter samples of type **Foo** (p. 1916).

7.101.3.1 Writing the Compile Function

The first thing to note is that we can ignore the filter expression, since we already know what the expression is. The second is that x is a parameter that can be changed. By using this information, the compile function is very easy to implement. Simply return the parameter string. This string will then be passed to the evaluate function every time a sample of this type is filtered.

Below is the entire **compile** (p. 416) function.

```
DDS_ReturnCode_t MyContentFilter::compile(
    void** new_compile_data, const char * /* expression */,
    const DDS_StringSeq& parameters,const DDS_TypeCode* /* type_code */,
    const char * /* type_class_name */,
    void * /* old_compile_data */) {
    *new_compile_data = (void*)DDS_String_dup(parameters[0]);
    return DDS_RETCODE_OK;
}
```

7.101.3.2 Writing the Evaluate Function

The next step is to implement the **evaluate** function. The evaluate function receives the parameter string with the actual value to test against. Thus the evaluate function must read the actual value from the parameter string before evaluating the expression. Below is the entire **evaluate** (p. 416) function.

```
DDS_Boolean MyContentFilter::evaluate(
    void* compile_data,const void* sample,const struct DDS_FilterSampleInfo * meta_data) {
    char *parameter = (char*)compile_data;
    DDS_Long x;
    Foo *foo_sample = (Foo*)sample;
    sscanf(parameter,"%d",&x);
    return (foo_sample->x > x ? DDS_BOOLEAN_FALSE : DDS_BOOLEAN_TRUE);
}
```

7.101.3.3 Writing the Finalize Function

The last function to write is the finalize function. It is safe to free all resources used by this particular instance of the custom content filter that is allocated in **compile**. Below is the entire **finalize** (p. 417) function.

```
void MyContentFilter::finalize(
    void* compile_data) {
    /* free parameter string from compile function */
    DDS_String_free((char *)compile_data);
}
```

7.101.3.4 Registering the Filter

Before the custom filter can be used, it must be registered with RTI Connext:

```
DDSDomainParticipant *myParticipant=NULL;
/* myParticipant = .... */
DDSContentFilter *myCustomFilter = new MyContentFilter();
if (myParticipant->register_contentfilter(
    (char*)"MyCustomFilter",
    myCustomFilter) != DDS_RETCODE_OK) {
    printf("Failed to register custom filter\n");
}
```

7.101.3.5 Unregistering the Filter

When the filter is no longer needed, it can be unregistered from RTI Connext:

```
DDSDomainParticipant *myParticipant = NULL;
/* myParticipant = .... */
DDSContentFilter *myCustomFilter =
    myParticipant->lookup_contentfilter((char*)"MyCustomFilter");
if (myCustomFilter != NULL) {
    if (myParticipant->unregister_contentfilter(
        (char*)"MyCustomFilter") != DDS_RETCODE_OK) {
        printf("Failed to unregister custom filter\n");
    }
    delete myCustomFilter;
}
```

7.102 Large Data Use Cases

Working with large data types.

Working with large data types.

7.102.1 Introduction

RTI Connext supports data types whose size exceeds the maximum message size of the underlying transports. A **DDSDataWriter** (p. 1589) will fragment data samples when required. Fragments are automatically reassembled at the receiving end.

Once all fragments of a sample have been received, the new sample is passed to the **DDSDataReader** (p. 1556) which can then make it available to the user. Note that the new sample is treated as a regular sample at that point and its availability depends on standard QoS settings such as **DDS_ResourceLimitsQosPolicy::max_samples** (p. 1311) and **DDS_KEEP_LAST_HISTORY_QOS** (p. 604).

The large data feature is fully supported by all DDS API's, so its use is mostly transparent. Some additional considerations apply as explained below.

7.102.2 Writing Large Data

In order to use the large data feature with the **DDS_RELIABLE_RELIABILITY_QOS** (p. 633) setting, the **DDSData↩ Writer** (p. 1589) must be configured as an asynchronous writer (**DDS_ASYNCHRONOUS_PUBLISH_MODE_QOS** (p. 629)) with associated **DDSFlowController** (p. 1736).

While the use of an asynchronous writer and flow controller is optional when using the **DDS_BEST_EFFORT_↩ RELIABILITY_QOS** (p. 633) setting, most large data use cases will benefit from the use of a flow controller to prevent flooding the network when fragments are being sent.

- **Set up writer** (p. 400)
- **Add flow control** (p. 397)

7.102.3 Receiving Large Data

Large data is supported by default and in most cases, no further changes are required.

The **DDS_DataReaderResourceLimitsQosPolicy** (p. 867) allows tuning the resources available to the **DDSData↩ Reader** (p. 1556) for reassembling fragmented large data.

- **Set up reader** (p. 403)

7.103 Request-Reply Examples

Examples on how to use the request-reply API .

Examples on how to use the request-reply API .

Request-Reply code examples.

7.103.1 Request-Reply Examples

Requesters and Repliers provide a way to use the Request-Reply communication pattern on top of the DDS entities. An application uses a Requester to send requests to a Replier; another application using a Replier receives a request and can send one or more replies for that request. The Requester that sent the request (and only that one) will receive the reply (or replies).

DDS Types

RTI Connext uses DDS data types for sending and receiving requests and replies. Valid types are those generated by the rtiddsgen code generator, the DDS built-in types, and `DynamicData`. Refer to the `Core Libraries User's Manual` and the following links for more information:

- Code Generator User's Manual,
- **Using the DDS built-in types** (p. 95),
- **Using DynamicData** (p. 103)

Set up

- **Create a DomainParticipant** (p. 395)
- **Create a requester** (p. 421)
- **Create a requester with parameters** (p. 421)
- **Create a replier** (p. 425)

Requester: sending requests and receiving replies

- **Basic Requester example** (p. 421)
- **Taking loaned samples** (p. 422)
- **Taking samples by copy** (p. 422)
- **Correlation between requests and replies** (p. 423)

Replier: receiving requests and sending replies

- **Basic Replier example** (p. 425)
- **SimpleReplier example** (p. 426)
- **Taking loaned samples** (p. 422)
- **Taking samples by copy** (p. 422)

Error handling

- **Error handling example** (p. 426)

Note

To use Request-Reply you need to build and link your application with the additional `rticonnextmsgcpp` library and include `ndds/ndds_requestreply_cpp.h`.

7.103.2 Requester Creation

- **Setting up a participant** (p. 395)
- Creating a Requester

```
using namespace connext;
// Note: error checking omitted for simplicity
// Create a DomainParticipant
DDS::DomainParticipant * participant =
    DDS::DomainParticipantFactory::get_instance()->create_participant(
        domain_id, DDS::PARTICIPANT_QOS_DEFAULT,
        NULL, DDS::STATUS_MASK_NONE);
// Create a Requester (if creation fails, the constructor throws)
Requester<Foo, Bar> * requester =
    new Requester<Foo, Bar>(participant, "TestService");
```

7.103.3 Creating a Requester with optional parameters

- **Setting up a participant** (p. 395)
- Creating a Requester with additional parameters

```
using namespace connext;
// Note: error checking omitted for simplicity
// Create a DomainParticipant
DDS::DomainParticipant * participant =
    DDS::DomainParticipantFactory::get_instance()->create_participant(
        domain_id, DDS::PARTICIPANT_QOS_DEFAULT,
        NULL, DDS::STATUS_MASK_NONE);
// Create a Requester with a QoS profile (located for example in
// USER_QOS_PROFILES.xml, in the current working directory)
RequesterParams requester_params(participant);
requester_params.service_name("TestService");
requester_params.qos_profile(
    "RequestReplyExampleProfiles", "RequesterExampleProfile");
Requester<Foo, Bar> * requester = new Requester<Foo, Bar>(requester_params);
```

See also

Requester Creation (p. 421)

Configuring Request-Reply QoS profiles (p. 427)

7.103.4 Basic Requester example

- **Requester Creation** (p. 421)
- **Creating a Requester with optional parameters** (p. 421)
- Basic Requester example

```
using namespace connext;
// Send request
WriteSample<Foo> request;
strcpy(request.data().message, "A Request");
requester->send_request(request);
// Create a reply Sample
Sample<Bar> reply;
// Receive reply (wait for it and get the sample)
bool received = requester->receive_reply(reply, MAX_WAIT);
if (received) {
    if(reply.info().valid_data) {
        std::cout << "Received reply: " << reply.data().message << std::endl;
    } else {
        std::cout << "Received invalid reply" << std::endl;
    }
} else {
    std::cout << "Reply not received" << std::endl;
}
```

- Basic Requester example using the **String built-in type** (p. 508)

```
using namespace connext;
// Create a Requester (this time we create it on the stack)
// The template parameter std::string maps to the String DDS built-in type
Requester<std::string, std::string> requester(participant, "TestServiceString");
// Send a request
requester.send_request("A string request");
// Receive reply
LoanedSamples<std::string> replies = requester.receive_replies(MAX_WAIT);
for (LoanedSamples<std::string>::iterator it = replies.begin();
     it != replies.end();
     ++it) {
    if (it->info().valid_data) {
        std::cout << "Received string reply: " << it->data() << std::endl;
    }
}
// When replies go out of scope, the loan is automatically returned
```

See also

Basic Replier example (p. 425)

SimpleReplier example (p. 426)

7.103.5 Taking loaned samples

- **Requester Creation** (p. 421)
- Get iterator to loaned replies (no copies)

```
using namespace connext;
// Get loaned samples
// Unlike receive_replies(), take_replies() doesn't block and returns an
// empty collection if there are no replies at that moment.
// (before that we can call requester->wait_for_replies)
LoanedSamples<Bar> replies = requester->take_replies();
// Iterate through the container of loaned samples.
// If there were no replies, the container is empty
for (LoanedSamples<Bar>::const_iterator it = replies.begin();
    it != replies.end();
    ++it) {
    if (it->info().valid_data) {
        std::cout << "Received reply: " << it->data().message << std::endl;
    }
}
// When replies go out of scope, the loan is automatically returned
```

See also

Basic Requester example (p. 421)

Basic Replier example (p. 425)

Taking samples by copy (p. 422)

7.103.6 Taking samples by copy

- **Requester Creation** (p. 421)
- Get copies of the replies:

```
using namespace connext;
// Get loaned samples
LoanedSamples<Bar> replies = requester->take_replies();
// Copy the contents using the STL-compliant iterator in LoanedSamples
std::list<Sample<Bar> > my_list;
std::copy(replies.begin(), replies.end(), std::back_inserter(my_list));
```

```
// We can copy to a container of Bar rather than Sample<Bar>,
// thanks to the implicit conversion from Sample<Bar> to Bar.
// If our application works with instances, we may have received
// invalid data (e.g. disposed instance). We can iterate through valid
// samples only using an iterator adapter.
std::vector<Bar> my_vector;
std::copy(
    make_valid_sample_iterator(replies.begin()),
    make_valid_sample_iterator(replies.end()),
    std::back_inserter(my_vector));
// When replies go out of scope, the loan is automatically returned
```

See also

Basic Replier example (p. 425)

Taking loaned samples (p. 422)

7.103.7 Correlating requests and replies

- **Requester Creation** (p. 421)
- Example 1) Waiting for a reply to a specific request

```
using namespace connext;
// Create requests
WriteSample<Foo> request1, request2;
strcpy(request1.data().message, "Request 1");
strcpy(request2.data().message, "Request 2");
// Send requests using WriteSample
requester->send_request(request1);
requester->send_request(request2);
// Wait for a reply to the second request
Sample<Bar> reply;
bool received = requester->wait_for_replies(
    1, MAX_WAIT, request2.identity());
if (!received) {
    std::cout << "Did not receive reply for request 2" << std::endl;
    return;
}
// Take that reply
requester->take_reply(reply, request2.identity());
// This postcondition should always be true
if (reply.related_identity() != request2.identity()) {
    throw std::runtime_error("postcondition failed");
}
if (reply.info().valid_data) {
    std::cout << "Received reply for request 2: "
              << reply.data().message << std::endl;
}
// Wait for a reply to the first request
received = requester->wait_for_replies(1, MAX_WAIT, request1.identity());
if (!received) {
    std::cout << "Did not receive reply for request 1" << std::endl;
    return;
}
// Take that reply
requester->take_reply(reply, request1.identity());
// This postcondition should always be true
if (reply.related_identity() != request1.identity()) {
    throw std::runtime_error("postcondition failed");
}
if (reply.info().valid_data) {
    std::cout << "Received reply for request 1: "
              << reply.data().message << std::endl;
}
```

- Example 2) Correlating a reply after receiving it

```
using namespace connext;
// Create requests
WriteSample<Foo> request1, request2;
strcpy(request1.data().message, "Request 1");
strcpy(request2.data().message, "Request 2");
```

```
// Send requests using WriteSample
requester->send_request(request1);
requester->send_request(request2);
// Wait for two replies. In this case we don't mind the
// reception order
bool received = requester->wait_for_replies(2, MAX_WAIT);
if (!received) {
    std::cout « "Replies not received" « std::endl;
    return;
}
LoanedSamples<Bar>::iterator it;
// Get all the replies
LoanedSamples<Bar> replies = requester->take_replies();
// Find the reply for request 1.
//
// We search using the STL find_if algorithm. The search range is all
// the elements in the replies container (from beginning to end).
// We use a predicate included in the Connext namespace that evaluates to
// true when a Sample's related_identity equals the identity that
// is passed to the constructor (request1's, in this case)
it = std::find_if(replies.begin(), replies.end(),
                  IsReplyRelatedPredicate<Bar>(request1.identity()));
if (it != replies.end()) {
    std::cout « "Received reply for request 1: "
              « it->data().message « std::endl;
}
// Find the reply for request 2
it = std::find_if(replies.begin(), replies.end(),
                  IsReplyRelatedPredicate<Bar>(request2.identity()));
if (it != replies.end()) {
    std::cout « "Received reply for request 2: "
              « it->data().message « std::endl;
}
// When replies go out of scope, the loan is automatically returned
```

See also

Basic Requester example (p. 421)

Basic Replier example (p. 425)

7.103.8 Basic Requester example using SampleRef

This example is similar to **Basic Requester example** (p. 421) except it uses a SampleRef instance instead of a Sample instance.

- Assuming the variables you want to use already exist:

```
Foo request_data;
DDS::WriteParams_t request_params;
Bar reply_data;
DDS::SampleInfo reply_info;
// ...
```

- You can use WriteSampleRef and SampleRef:

```
using namespace connext;
// Create a request WriteSampleRef
WriteSampleRef<Foo> request;
request.set_data(request_data);
request.set_info(request_params);
// Send request
requester->send_request(request);
// Create a reply SampleRef
SampleRef<Bar> reply;
reply.set_data(reply_data);
reply.set_info(reply_info);
// Receive reply
bool received = requester->receive_reply(reply, MAX_WAIT);
if (received) {
    if(reply_info.valid_data) {
        std::cout « "Received reply: " « reply_data.message « std::endl;
```

```
        } else {
            std::cout « "Received invalid reply" « std::endl;
        }
    } else {
        std::cout « "Reply not received" « std::endl;
    }
```

See also

connext::SampleRef (p. 2191)

Basic Requester example (p. 421)

7.103.9 Creating a Replier

- **Setting up a participant** (p. 395)
- Create a Replier

```
using namespace connext;
// Note: error checking omitted for simplicity
// Create a DomainParticipant
DDS::DomainParticipant * participant =
    DDS::DomainParticipantFactory::get_instance()->create_participant(
        domain_id, DDS::PARTICIPANT_QOS_DEFAULT,
        NULL, DDS::STATUS_MASK_NONE);
// Create a Replier (if creation fails, the constructor throws exception)
Replier<Foo, Bar> * replier =
    new Replier<Foo, Bar>(participant, "TestService");
```

7.103.10 Basic Replier example

- **Creating a Replier** (p. 425)
- Basic Replier example

```
using namespace connext;
Sample<Foo> request;
// Receive one request
bool received = replier->receive_request(request, MAX_WAIT);
if (!received) {
    std::cout « "Requests not received" « std::endl;
    return false;
}
// A WriteSample automatically initializes and finalizes
// the data using the TypeSupport (in this case BarTypeSupport)
WriteSample<Bar> reply;
if (request.info().valid_data) {
    sprintf(reply.data().message, "Reply for %s", request.data().message);
    // Send a reply for that request
    replier->send_reply(reply, request.identity());
    // Note: a replier can send more than one reply for the same request
}
```

See also

Basic Requester example (p. 421)

7.103.11 SimpleReplier example

- Implement a listener

```
class MySimpleReplierListener : public SimpleReplierListener<Foo, Bar>
{
public:
    Bar * on_request_available(SampleRef<Foo> request)
    {
        if (!request.info().valid_data) {
            // In this example we don't reply to invalid data samples.
            // In other cases, it could make sense to reply
            // to a disposed instance, for example.
            return NULL;
        }
        sprintf(_reply.data().message, "Simple reply for %s",
                request.data().message);
        return &_reply.data();
    }
    void return_loan(Bar * reply)
    {
        // nothing to do
    }
private:
    // By using a WriteSample (instead of just Bar), the sample is
    // automatically initialized and released.
    WriteSample<Bar> _reply;
};
```

- And create a SimpleReplier with the listener

```
    using namespace connext;
    // Note: error checking omitted for simplicity
    MySimpleReplierListener * listener = new MySimpleReplierListener();
    SimpleReplier<Foo, Bar> * replier = new SimpleReplier<Foo, Bar>(
        participant, "TestService", *listener);
    // After creation the SimpleReplier is already active and may call
    // the listener
```

- This next example shows how to implement a listener for a SimpleReplier using the std::string type

```
class MyStringSimpleReplierListener :
    public SimpleReplierListener<std::string, std::string>
{
public:
    // Note that the actual received type is const char*,
    // because internally, the middleware uses char*
    std::string * on_request_available(SampleRef<const char*> request)
    {
        if (!request.info().valid_data) {
            return NULL;
        }
        _reply = std::string("Simple string reply for ") + request.data();
        return &_reply;
    }
    void return_loan(std::string * reply)
    {
        // nothing to do
    }
private:
    std::string _reply;
};
```

See also

Basic Requester example (p. 421)

7.103.12 Error handling example

- Catching an exception from the request-reply API

```
    using namespace connext;
    // All the operations in the request-reply API
    // throw exceptions in case of error.
    // You can also set the verbosity level to warning for
    // additional information
    NDDSConfigLogger::get_instance()->set_verbosity(
```

```
    NDDS_CONFIG_LOG_VERBOSITY_WARNING);
//
// For example, this replier is sending a reply for a nonexistent request
//
WriteSample<Bar> reply;
DDS::SampleIdentity_t invalid_request_id;
try {
    replier->send_reply(reply, invalid_request_id);
} catch (const BadParameterException& ex) {
    std::cout << "Exception while sending reply: "
              << ex.what() << std::endl;
}
// Exceptions inherit from std::runtime_error or std::logic_error,
// so you can have a more general catch statement:
//
// This piece of code produces the same result as the previous one:
//
try {
    replier->send_reply(reply, invalid_request_id);
} catch (const std::exception& ex) {
    std::cout << "Exception while sending request: "
              << ex.what() << std::endl;
}
```

7.103.13 Configuring Request-Reply QoS profiles

If you do not specify your own QoS parameters (in **connext::RequesterParams** (p. 2178) and **connext::ReplierParams** (p. 2152)), a **connext::Requester** (p. 2157) and **connext::Replier** (p. 2139) are created using a default configuration defined in the built-in profile **DDS_PROFILE_PATTERN_RPC** (p. 697). You use this built-in profile as the base for your own profiles:

```
<?xml version="1.0" encoding="UTF-8"?>
<!--
Description
Example XML QoS Profile for HelloWorld


(c) Copyright, Real-Time Innovations, 2012.  All rights reserved.
RTI grants Licensee a license to use, modify, compile, and create derivative
works of the software solely for use with RTI Connext DDS. Licensee may
redistribute copies of the software provided that all such copies are
subject to this license. The software is provided "as is", with no warranty
of any type, including any warranty for fitness for any purpose. RTI is
under no obligation to maintain or support the software. RTI shall not be
liable for any incidental or consequential damages arising out of the use
or inability to use the software.


The QoS configuration of the DDS entities in the generated example is loaded
from this file.


This file is used only when it is in the current working directory, the
environment variable NDDS_QOS_PROFILES is defined and points to this file, or
if the filename is specified in code.


QoS profiles are a tool for pre-specifying behavior of DDS entities (e.g.
DomainParticipants, DataReaders, and DataWriters).  For Domain Participants, this
means primarily configuring platform resources (e.g. UDP buffers sizes). For
DataReaders and DataWriters, it is aligning their behavior with the required
design patterns and ensuring the QoS for DataWriters and DataReaders are
compatible at runtime.


The profiles in this file use inheritance from built-in QoS Profiles and
composition from "QoS Snippets", which are provided in this product, to create
Four example profiles.  One for a DomainParticipant, and three additional profiles
which specify DataReader and DataWriter behavior for common communication design
Patterns.


The profiles will require optimization for networks that are highly variable in
throughput, latency, and/or sample loss, or have other unique characteristics.
Applications with special resource, latency or throughput requirements will also
require optimized profiles.
```

```
A collection of built-in profiles can be found in the
BuiltinProfiles.documentationONLY.xml file located in the
$NDDSHOME/resource/xml/ directory.


Any of these "base" QoS configurations may be employed by referring to them by
the name shown in the BuiltinProfiles.documentationONLY.xml file.


There is also a library that contains a collection of
QoS Snippets that set some specific features or configurations. They are
also located in the BuiltinProfiles.documentationONLY.xml file.
https://community.rti.com/best-practices/qos-profile-inheritance-and-composition-guidance


The file BuiltinProfiles.documentationONLY.xml is not intended to
be changed by users. To modify any of the values in a built-in profile,
the example in this file which inherits from a built-in profile should be
used, with changes applied as needed.


The "Configuring QoS with XML" chapter in the RTI Connext DDS Core Libraries User's Manual
has more information about managing QoS using XML QoS Profiles.
-->
<dds xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"

       xsi:noNamespaceSchemaLocation="file:////rti/jenkins/workspace/connextdds_htmldocs_support_connextdds-doc_7.7.0/build/nddsg
    <!-- A QoS library is a named set of QoS profiles.  This QoS Library contains
         the QoS profiles which may be used in the generated example
    -->
    <qos_library name="HelloWorld_Library">
        <!-- A QoS profile groups a set of related QoS for one or more DDS Entities -->
        <!-- The following design-pattern oriented profiles are intended only for DataReader and DataWriter DDS
       Entities -->
        <qos_profile name="Aperiodic_Reliable_Profile" base_name="BuiltinQosLib::Generic.Common">
            <!-- Aperiodic Reliable behavior is suitable for Command, Response, Event or Alarm use-cases that
       require reliable data delivery. Reliability is the protocol by which samples that are lost for any reason
       are retransmitted to reliable DataReaders. In this profile, samples are not retained for late joining
       DataReaders. Reliability begins when a DataWriter matches a DataReader at discovery.
An average DataWriter send frequency of > 500 ms is appropriate for this profile.
            -->
            <base_name>
                <!-- Makes DataReader and DataWriter reliable (BEST_EFFORT DataReaders allowed).
                     DataWriter will block up to 5 seconds if cache becomes full of unacknowledged
                     samples, after which a write() call will fail if cache is still full -->
                <element>BuiltinQosSnippetLib::QosPolicy.Reliability.Reliable</element>
                <!-- DataWriters will use all available cache for reliability retransmission,
                     DataReaders use all available cache to save samples until removed -->
                <element>BuiltinQosSnippetLib::BuiltinQosSnippetLib::QosPolicy.History.KeepAll</element>
                <!-- Optimize reliability timing for a stable, "normal" network,
                     allowing DataWriter cache to grow from default of 32 samples to 40,
                     DataReader cache to grow from 32 samples to 256. Datawriter issues heartbeats for
                     acknowledgements every 200 ms while there are unacknowledged samples
                     and will issue up to 500 periodic retries before reliability fails -->
                <element>BuiltinQosSnippetLib::Optimization.ReliabilityProtocol.Common</element>
            </base_name>
            <!-- Name Datawriters and Datareaders by use-case -->
            <datawriter_qos>
                <publication_name>
                    <name>HelloWorld_AR_DataWriter</name>
                </publication_name>
            </datawriter_qos>
            <datareader_qos>
                <subscription_name>
                    <name>HelloWorld_AR_DataReader</name>
                </subscription_name>
            </datareader_qos>
            <domain_participant_qos>
                <!--
                The participant name, if it is set, will be displayed in the
                RTI tools, making it easier for you to tell one
                application from another when you're debugging.
                -->
                <participant_name>
                    <name>HelloWorld_AR_Participant</name>
                    <role_name>AR_HelloWorld_AR_ParticipantRole</role_name>
                </participant_name>
            </domain_participant_qos>
        </qos_profile>
```

```
 <qos_profile name="Periodic_Best_Effort_Profile" base_name="BuiltinQosLib::Generic.Common">
     <!-- Periodic BEST_EFFORT behavior is suitable for samples that update frequently enough that
recovery
          of lost or out of order samples is not required. -->
     <!-- Datawriters will not allow Reliable DataReaders. DataWriter only needs history of 1 since
          there are no retransmissions with BEST_EFFORT -->
     <base_name>
         <element>BuiltinQosSnippetLib::QosPolicy.Reliability.BestEffort</element>
         <element>BuiltinQosSnippetLib::QosPolicy.History.KeepLast_1</element>
     </base_name>
     <!-- Name Datawriters and Datareaders by use-case -->
     <datawriter_qos>
         <publication_name>
             <name>HelloWorld_PBE_DataWriter</name>
         </publication_name>
     </datawriter_qos>
     <datareader_qos>
         <subscription_name>
             <name>HelloWorld_PBE_DataReader</name>
         </subscription_name>
         <!-- Increase DataReader cache to 100 samples per instance for incoming data to prevent
overwriting -->
         <history>
             <kind>KEEP_LAST_HISTORY_QOS</kind>
             <depth>100</depth>
         </history>
     </datareader_qos>
     <domain_participant_qos>
         <!--
         The participant name, if it is set, will be displayed in the
         RTI tools, making it easier for you to tell one
         application from another when you're debugging.
         -->
         <participant_name>
             <name>HelloWorld_PBE_Participant</name>
             <role_name>HelloWorld_PBE_ParticipantRole</role_name>
         </participant_name>
     </domain_participant_qos>
 </qos_profile>
 <qos_profile name="Last_Value_Profile" base_name="Aperiodic_Reliable_Profile">
     <!-- Last Value behavior is suitable for state data updates or resolving race conditions at
discovery.
         This profile extends the Aperiodic_Reliable_Profile to provide the last update of each instance
to late joining DataReaders when it is discovered,
         without having to query the DataWriter   -->
     <!-- DataWriter retains samples (per instance) for late joining DataReaders,
          DataReaders request last values upon discovery of DataWriters -->
     <base_name>
         <element>BuiltinQosSnippetLib::QosPolicy.Durability.TransientLocal</element>
     </base_name>
     <!-- Name Datawriters and Datareaders by use-case -->
     <datawriter_qos>
         <publication_name>
             <name>HelloWorld_LV_DataWriter</name>
         </publication_name>
         <!-- Have DataWriters keep only the most recently updated sample for delivery
              to late joining DataReaders, still use all available cache for reliability -->
         <durability>
             <writer_depth>1</writer_depth>
         </durability>
     </datawriter_qos>
     <datareader_qos>
         <subscription_name>
             <name>HelloWorld_LV_DataReader</name>
         </subscription_name>
         <!-- Keep up to 256 samples per instance for incoming samples before overwriting the oldest -->
         <history>
             <kind>KEEP_LAST_HISTORY_QOS</kind>
             <depth>256</depth>
         </history>
     </datareader_qos>
 </qos_profile>

 <!-- DDS DomainParticipant QoS specifies discovery behavior and platform resource
      values. These settings are appropriate for Windows or Linux server platforms and
      are intended only for DDS DomainParticipant entities -->
 <qos_profile name="Participant_Profile" base_name="BuiltinQosLib::Generic.Common">
     <domain_participant_qos>
         <!-- To have a unique name for each DomainParticipant, the name may be set in code for the
DomainParticipant QoS, or in an environment variable referenced in the name tag.  -->
         <participant_name>
```

```
                <name>HelloWorld_LV_Participant</name>
            </participant_name>
            <transport_builtin>
                <udpv4>
                    <!-- For high throughput implementations or cases where there are
                         many DDS Entities to be discovered, larger UDP buffer sizes
                         are essential for discovery, latency, and/or throughput performance,
                         especially for non-RTOS operating systems like Linux or Windows.
                         UDP receive socket buffers are especially important. Linux, for
                         example, has small default values and should be configured to support buffers equal
      to or greater than these QoS values. Reference
      https://community.rti.com/howto/improve-rti-connext-dds-network-performance-linux for additional
          details.  Windows does not usually require special configuration.
                    <send_socket_buffer_size>10485760</send_socket_buffer_size>
                    <recv_socket_buffer_size>2097152</recv_socket_buffer_size> -->
                    <!-- Connext will try to use all available networks on a given host.
                         An allow list (white list) helps Connext use only the specified
                         interface(s) and prevents duplicate sending of data. For more information:

      https://community.rti.com/kb/how-do-i-restrict-and-prioritize-interfaces-rti-connext-use
                    <allow_interfaces_list>
                        <element>192.168.0.*</element>
                    </allow_interfaces_list>
                    -->
                </udpv4>
            </transport_builtin>
        </domain_participant_qos>
      </qos_profile>
    <!--  This profile is provided for reference and can be uncommented
    for debug purposes. Should additional logging information be required
    or requested by RTI Support the values should be changed from the
    defaults specified here


      <qos_profile name="Domain_Participant_Factory_Qos">
          <participant_factory_qos is_default_participant_factory_qos="true">
              <logging>
          <category>ALL</category>
          <output_file/>
          <print_format>DEFAULT</print_format>
          <verbosity>ERROR</verbosity>
      </logging>
         </participant_factory_qos>
      </qos_profile>
-->
    </qos_library>
</dds>
```

The example **Creating a Requester with optional parameters** (p. 421) shows how to create a **connext::Requester** (p. 2157) using this profile.

See also

Creating a Requester with optional parameters (p. 421)

Configuring QoS Profiles with XML (p. 390)

7.104 Documentation Roadmap

This section contains a roadmap for the new user with pointers on what to read first.

If you are new to RTI Connext, we recommend starting in the following order:

- See the `Getting Started Guide`. This document provides download and installation instructions. It also lays out the core value and concepts behind the product and takes you step-by-step through the creation of a simple example application.

- The `User's Manual` describes the features of the product and how to use them. It is organized around the structure of the DDS APIs and certain common high-level tasks.

- The documentation in the **RTI Connext DDS API Reference** (p. 434) provides an overview of API classes and modules for the DDS data-centric publish-subscribe (DCPS) package from a programmer's perspective. Start by reading the documentation on the main page.
- After reading the high level module documentation, look at the **Publication Example** (p. 392) and **Subscription Example** (p. 393) for step-by-step examples of creating a publication and subscription. These are hyperlinked code snippets to the full API documentation, and provide a good place to begin learning the APIs.

- Next, work through your own application using the example code files generated by rtiddsgen. See the `Code Generator User's Manual`.
- To integrate similar code into your own application and build system, you will likely need to refer to the `Platform Notes`.

7.105 Conventions

This section describes the conventions used in the API documentation.

7.105.1 Unsupported Features

[Not supported (optional)] This note means that the optional feature from the DDS specification is not supported in the current release.

7.105.2 API Naming Conventions

7.105.2.1 Structure & Class Names

RTI Data Distribution Service 4 makes a distinction between *value types* and *interface types*. Value types are types such as primitives, enumerations, strings, and structures whose identity and equality are determined solely by explicit state. Interface types are those abstract opaque data types that conceptually have an identity apart from their explicit state. Examples include all of the **DDSEntity** (p. 1731) subtypes, the **DDSCondition** (p. 1544) subtypes, and **DDSWaitSet** (p. 1897). Instances of value types are frequently transitory and are declared on the stack. Instances of interface types typically have longer lifecycles, are accessible by pointer only, and may be managed by a factory object.

Value and interface types are distinguished by their names: value types have names beginning with "DDS_" (i.e. with an underscore); interface types have names beginning with "DDS" (i.e. with no underscore). Another way to think of it: C-style types – structures, enumerations, etc. – have names beginning with "DDS_"; C++ classes have names beginning with "DDS."

7.105.3 API Documentation Terms

In the API documentation, the term module refers to a logical grouping of documentation and elements in the API.

At this time, typedefs that occur in the API, such as **DDS_ReturnCode_t** (p. 532) do not show up in the compound list or indices. This is a known limitation in the generated HTML.

7.105.4 Stereotypes

Commonly used stereotypes in the API documentation include the following.

7.105.4.1 Basic Qos

- *<<**basic**>> (p. 201)*
  - A QoS that Connext considers Basic.
  - See **Basic QoS** (p. 201) for a full list.

7.105.4.2 Extensions

- *<<**spec**>>* (p. *209)*
  - Part of a DDS standard specification.
  - See **DDS Specification QoS** (p. 209) for a full list of the QoS policies and fields that are part of a DDS standard specification.
- *<<**extension**>>* (p. *432)*
  - An RTI Connext product extension to the DDS standard specification.
  - The extension APIs complement the standard APIs specified by the OMG DDS specification. They are provided to improve product usability and enable access to product-specific features such as pluggable transports.
  - See **DDS Extension QoS** (p. 213) for a full list of QoS policies and fields that are extensions.

7.105.4.3 Experimental

- *<<**experimental**>> (p. 432)*
  - RTI Connext experimental features are used to evaluate new features and get user feedback.
  - These features are not guaranteed to be fully supported and might be implemented only of some of the programming languages supported bt RTI Connext
  - The functional APIs corresponding to experimental features can be distinguished from other APIs by the suffix '_exp'.
  - Experimental features may or may not appear in future product releases.
  - The name of the experimental features APIs will change if they become officially supported. At the very least the suffix '_exp' will be removed.
  - Experimental features should not be used in production.

7.105.4.4 Types

- *<<**interface**>> (p. 433)*
  - Pure interface type with *no state*.
  - Languages such as Java natively support the concept of an *interface* type, which is a collection of method signatures devoid of any dynamic state.
  - In C++, this is achieved via a class with all *pure virtual* methods and devoid of any instance variables (ie no dynamic state).
  - Interfaces are generally organized into a type hierarchy. Static typecasting along the interface type hierarchy is "safe" for valid objects.
- *<<**generic**>> (p. 433)*
  - A *generic* type is a *skeleton* class written in terms of generic parameters. Type-specific instantiations of such types are conventionally referred to in this documentation in terms of the hypothetical type "Foo"; for example: **FooSeq** (p. 1965), FooDataType, **FooDataWriter** (p. 1944), and **FooDataReader** (p. 1917).
  - For portability and efficiency, we implement generics using C preprocessor macros, rather than using C++ templates.
  - A *generic* type interface is declared via a #define macro.
  - Concrete types are generated from the generic type statically at compile time. The implementation of the concrete types is provided via the generic macros which can then be compiled as normal C or C++ code.
- *<<**singleton**>> (p. 433)*
  - Singleton class. There is a single instance of the class.
  - Generally acccessed via a `get_instance()` static method.

7.105.4.5 Method Parameters

- *<<**in**>> (p. 433)*
  - An *input* parameter.
- *<<**out**>> (p. 433)*
  - An *output* parameter.
- *<<**inout**>> (p. 433)*
  - An *input* and *output* parameter.

7.106 Using DDS:: Namespace

This section describes the C++ namespace support in the DDS API.

7.106.1 DDS Namespace Support

In this documentation, all C++ classes, value types, interface types and constants have names beginning with either **"DDS_"** or **"DDS"**. Alternatively, **DDS** namespace can also be used to refer to all these classes, types or constant.

All the C++ API that begins with either **"DDS_"** or **"DDS"** can be replaced with its namespace equivalent. For example, **DDSDomainParticipant (**p. **1619)** has a namespace equivalent of **DDS::DomainParticipant**, and **DDS_Domain↩ ParticipantQos (**p. **971)** has a namespace equivalent of **DDS::DomainParticipantQos**.

In order to use the **DDS** namespace, an additional header file, **ndds_namespace_cpp.h** , will need to be included in your source file:

```
#include "ndds/ndds_cpp.h"
#include "ndds/ndds_namespace_cpp.h"
DDS::DomainParticipant *participant = NULL;
DDS::DomainParticipantQos participant_qos;
DDS::DomainParticipantListener *listener = NULL;
DDS::StatusKind status_kind = DDS::INCONSISTENT_TOPIC_STATUS;
DDS::Long counter = 0L;
```

If the namespace header file is not included in the source file, DDS namespace cannot be used in the RTI Connext API.

7.106.2 DDS Namespace and Primitive Types

By default, **DDS** namespace support for primitive types are included. With **DDS** namespace support, the difference between DDS types and native types can just be the capitalization in some cases:

```
#include "ndds/ndds_cpp.h"
#include "ndds/ndds_namespace_cpp.h"
using namespace DDS;
Long ddsCounter = 0L;
long nativeCounter = 0L;
```

If you want to exclude the DDS namespace support for primitive types, you can define **NDDS_EXCLUDE_PRIMITIVE_↩ TYPES_FROM_NAMESPACE** in you application before including the namespace header file. **DDS** namespace support for primitive types will then be excluded:

```
#include "ndds/ndds_cpp.h"
#define NDDS_EXCLUDE_PRIMITIVE_TYPES_FROM_NAMESPACE
#include "ndds/ndds_namespace_cpp.h"
using namespace DDS;
DDS_Long ddsCounter = 0;
long nativeCounter = 0;
```

For the rest of the documentation, the **DDS** prefix is used for all class /types/constants names. However, all the API with the **DDS** prefix can be replaced with **DDS** namespace instead if the namespace header file is included.

7.107 RTI Connext DDS API Reference

RTI Connext modules following the DDS module definitions.

Modules

- **Domain Module**

  *Contains the **DDSDomainParticipant** (p. 1619) class that acts as an entrypoint of RTI Connext and acts as a factory for many of the classes. The **DDSDomainParticipant** (p. 1619) also acts as a container for the other objects that make up RTI Connext.*
- **Topic Module**

  *Contains the **DDSTopic** (p. 1885), **DDSContentFilteredTopic** (p. 1551), and **DDSMultiTopic** (p. 1797) classes, the **DDSTopicListener** (p. 1894) interface, and more generally, all that is needed by an application to define **DDSTopic** (p. 1885) objects and attach QoS policies to them.*
- **Publication Module**

  *Contains the **DDSFlowController** (p. 1736), **DDSPublisher** (p. 1818), and **DDSDataWriter** (p. 1589) classes as well as the **DDSPublisherListener** (p. 1839) and **DDSDataWriterListener** (p. 1612) interfaces, and more generally, all that is needed on the publication side.*
- **Subscription Module**

  *Contains the **DDSSubscriber** (p. 1860), **DDSDataReader** (p. 1556), **DDSReadCondition** (p. 1842), **DDSQuery↩ Condition** (p. 1841), and **DDSTopicQuery** (p. 1895) classes, as well as the **DDSSubscriberListener** (p. 1881) and **DDSDataReaderListener** (p. 1583) interfaces, and more generally, all that is needed on the subscription side.*
- **Infrastructure Module**

  Defines the abstract classes and the interfaces that are refined by the other modules. Contains common definitions such as return codes, status values, and QoS policies.
- **Transports**

  APIs related to RTI Connext pluggable transports.
- **Queries and Filters Syntax**
- **Logging and Version**

  APIs of troubleshooting utilities and APIs designed to configure the overall behavior of RTI Connext.
- **General Utilities and Compliance Configuration**

  API of general utilities used in the RTI Connext distribution.
- **Observability Framework**

  RTI Connext Observability Framework modules and APIs.
- **Durability and Persistence**

  APIs related to RTI Connext Durability and Persistence.
- **System Properties**

  System Properties.
- **Configuring QoS Profiles with XML**

  APIs related to XML QoS Profiles.
- **Additional RTI Connext Communication Patterns**

  Extensions to the RTI Connext publish-subscribe functionality.

7.107.1 Detailed Description

RTI Connext modules following the DDS module definitions.

7.107.2 Overview

Information flows with the aid of the following constructs: **DDSPublisher** (p. 1818) and **DDSDataWriter** (p. 1589) on the sending side, **DDSSubscriber** (p. 1860) and **DDSDataReader** (p. 1556) on the receiving side.

- A **DDSPublisher** (p. 1818) is an object responsible for data distribution. It may publish data of different data types. A TDataWriter acts as a *typed* (i.e. each **DDSDataWriter** (p. 1589) object is dedicated to one application data type) accessor to a publisher. A **DDSDataWriter** (p. 1589) is the object the application must use to communicate to a publisher the existence and value of data objects of a given type. When data object values have been communicated to the publisher through the appropriate data-writer, it is the publisher's responsibility to perform the distribution (the publisher will do this according to its own QoS, or the QoS attached to the corresponding data-writer). A *publication* is defined by the association of a data-writer to a publisher. This association expresses the intent of the application to publish the data described by the data-writer in the context provided by the publisher.
- A **DDSSubscriber** (p. 1860) is an object responsible for receiving published data and making it available (according to the Subscriber's QoS) to the receiving application. It may receive and dispatch data of different specified types. To access the received data, the application must use a *typed* TDataReader attached to the subscriber. Thus, a *subscription* is defined by the association of a data-reader with a subscriber. This association expresses the intent of the application to subscribe to the data described by the data-reader in the context provided by the subscriber.

DDSTopic (p. 1885) objects conceptually fit between publications and subscriptions. Publications must be known in such a way that subscriptions can refer to them unambiguously. A **DDSTopic** (p. 1885) is meant to fulfill that purpose: it associates a name (unique in the domain i.e. the set of applications that are communicating with each other), a data type, and QoS related to the data itself. In addition to the topic QoS, the QoS of the **DDSDataWriter** (p. 1589) associated with that Topic and the QoS of the **DDSPublisher** (p. 1818) associated to the **DDSDataWriter** (p. 1589) control the behavior on the publisher's side, while the corresponding **DDSTopic** (p. 1885), **DDSDataReader** (p. 1556) and **DDSSubscriber** (p. 1860) QoS control the behavior on the subscriber's side.

When an application wishes to publish data of a given type, it must create a **DDSPublisher** (p. 1818) (or reuse an already created one) and a **DDSDataWriter** (p. 1589) with all the characteristics of the desired publication. Similarly, when an application wishes to receive data, it must create a **DDSSubscriber** (p. 1860) (or reuse an already created one) and a **DDSDataReader** (p. 1556) to define the subscription.

7.107.3 Conceptual Model

The overall conceptual model is shown below.

Notice that all the main communication objects (the specializations of Entity) follow unified patterns of:

- Supporting QoS (made up of several QosPolicy); QoS provides a generic mechanism for the application to control the behavior of the Service and tailor it to its needs. Each **DDSEntity** (p. 1731) supports its own specialized kind of QoS policies (see **All QoS Policies** (p. 549)).
- Accepting a **DDSListener** (p. 1794); listeners provide a generic mechanism for the middleware to notify the application of relevant asynchronous events, such as arrival of data corresponding to a subscription, violation of a QoS setting, etc. Each **DDSEntity** (p. 1731) supports its own specialized kind of listener. Listeners are related to changes in status conditions (see **Status Kinds** (p. 533)).

  Note that only one Listener per entity is allowed (instead of a list of them). The reason for that choice is that this allows a much simpler (and, thus, more efficient) implementation as far as the middleware is concerned. Moreover, if it were required, the application could easily implement a listener that, when triggered, triggers in return attached 'sub-listeners'.

- Accepting a **DDSStatusCondition** (p. 1846) (and a set of **DDSReadCondition** (p. 1842) objects for the **DDSDataReader** (p. 1556)); conditions (in conjunction with **DDSWaitSet** (p. 1897) objects) provide support for an alternate communication style between the middleware and the application (i.e., wait-based rather than notification-based).

All DCPS entities are attached to a **DDSDomainParticipant** (p. 1619). A domain participant represents the local membership of the application in a domain. A *domain* is a distributed concept that links all the applications able to communicate with each other. It represents a communication plane: only the publishers and the subscribers attached to the same domain may interact.

DDSDomainEntity (p. 1618) is an intermediate object whose only purpose is to state that a DomainParticipant cannot contain other domain participants.

At the DCPS level, data types represent information that is sent atomically. For performance reasons, only plain data structures are handled by this level.

By default, each data modification is propagated individually, independently, and uncorrelated with other modifications. However, an application may request that several modifications be sent as a whole and interpreted as such at the recipient side. This functionality is offered on a Publisher/Subscriber basis. That is, these relationships can only be specified among **DDSDataWriter** (p. 1589) objects attached to the same **DDSPublisher** (p. 1818) and retrieved among **DDSDataReader** (p. 1556) objects attached to the same **DDSSubscriber** (p. 1860).

By definition, a **DDSTopic** (p. 1885) corresponds to a single data type. However, several topics may refer to the same data type. Therefore, a **DDSTopic** (p. 1885) identifies data of a single type, ranging from one single instance to a whole collection of instances of that given type. This is shown below for the hypothetical data type **Foo** (p. 1916).

In case a set of instances is gathered under the same topic, different instances must be distinguishable. This is achieved by means of the values of some data fields that form the **key** to that data set. The *key description* (i.e., the list of data fields whose value forms the key) has to be indicated to the middleware. The rule is simple: *different data samples with the same key value represent successive values for the same instance, while different data samples with different key values represent different instances*. If no key is provided, the data set associated with the **DDSTopic** (p. 1885) is restricted to a *single instance*.

Topics need to be known by the middleware and potentially propagated. Topic objects are created using the create operations provided by **DDSDomainParticipant** (p. 1619).

The interaction style is straightforward on the publisher's side: when the application decides that it wants to make data available for publication, it calls the appropriate operation on the related **DDSDataWriter** (p. 1589) (this, in turn, will trigger its **DDSPublisher** (p. 1818)).

On the subscriber's side however, there are more choices: relevant information may arrive when the application is busy doing something else or when the application is just waiting for that information. Therefore, depending on the way the application is designed, asynchronous notifications or synchronous access may be more appropriate. Both interaction modes are allowed, a **DDSListener** (p. 1794) is used to provide a callback for synchronous access and a **DDSWaitSet** (p. 1897) associated with one or several **DDSCondition** (p. 1544) objects provides asynchronous data access.

The same synchronous and asynchronous interaction modes can also be used to access changes that affect the middleware communication status (see **Status Kinds** (p. 533)). For instance, this may occur when the middleware asynchronously detects an inconsistency. In addition, other middleware information that may be relevant to the application (such as the list of the existing topics) is made available by means of **built-in topics** (p. 61) that the application can access as plain application data, using built-in data-readers.

7.107.4 Modules

DCPS consists of five modules:

- **Infrastructure module** (p. 167) defines the abstract classes and the interfaces that are refined by the other modules. It also provides support for the two interaction styles (notification-based and wait-based) with the middleware.
- **Domain module** (p. 41) contains the **DDSDomainParticipant** (p. 1619) class that acts as an entrypoint of the Service and acts as a factory for many of the classes. The **DDSDomainParticipant** (p. 1619) also acts as a container for the other objects that make up the Service.
- **Topic module** (p. 63) contains the **DDSTopic** (p. 1885) class, the **DDSTopicListener** (p. 1894) interface, and more generally, all that is needed by the application to define **DDSTopic** (p. 1885) objects and attach QoS policies to them.
- **Publication module** (p. 107) contains the **DDSPublisher** (p. 1818) and **DDSDataWriter** (p. 1589) classes as well as the **DDSPublisherListener** (p. 1839) and **DDSDataWriterListener** (p. 1612) interfaces, and more generally, all that is needed on the publication side.
- **Subscription module** (p. 128) contains the **DDSSubscriber** (p. 1860), **DDSDataReader** (p. 1556), **DDSReadCondition** (p. 1842), and **DDSQueryCondition** (p. 1841) classes, as well as the **DDSSubscriberListener** (p. 1881) and **DDSDataReaderListener** (p. 1583) interfaces, and more generally, all that is needed on the subscription side.

7.108 Additional RTI Connext Communication Patterns

Extensions to the RTI Connext publish-subscribe functionality.

Modules

- **Request-Reply Pattern**

  Support for the request-reply communication pattern.
- **Infrastructure**

  Infrastructure types for Request-Reply.
- **Utilities**

  Utilities for the RTI Connext Messaging module.

7.108.1 Detailed Description

Extensions to the RTI Connext publish-subscribe functionality.

This section includes the **Request-Reply API** (p. 191).

7.109 Programming How-To's

These "How To"s illustrate how to apply RTI Connext APIs to common use cases.

Modules

- **Publication Example**

  A data publication example.
- **Subscription Example**

  A data subscription example.
- **Participant Use Cases**

  Working with domain participants.
- **Topic Use Cases**

  Working with topics.
- **FlowController Use Cases**

  Working with flow controllers.
- **Publisher Use Cases**

  Working with publishers.
- **DataWriter Use Cases**

  Working with data writers.
- **Subscriber Use Cases**

  Working with subscribers.
- **DataReader Use Cases**

  Working with data readers.
- **Entity Use Cases**

  Working with entities.
- **Waitset Use Cases**

  Using wait-sets and conditions.
- **Transport Use Cases**

  Working with pluggable transports.
- **Filter Use Cases**

  Working with data filters.
- **Creating Custom Content Filters**

  Working with custom content filters.
- **Large Data Use Cases**

  Working with large data types.
- **Request-Reply Examples**

  Examples on how to use the request-reply API .

7.109.1 Detailed Description

These "How To"s illustrate how to apply RTI Connext APIs to common use cases.

These are a good starting point to familiarize yourself with DDS. You can use these code fragments as "templates" for writing your own code.

7.110 Interface

Abstraction of a Transport Plugin network interface.

Classes

- struct **NDDS_Transport_Interface_t**

  Storage for the description of a network interface used by a Transport Plugin.

Enumerations

- enum **NDDS_Transport_Interface_Status_t** {
  NDDS_TRANSPORT_INTERFACE_OFF = 0 ,
  NDDS_TRANSPORT_INTERFACE_ON = 1 }

  Interface status.

7.110.1 Detailed Description

Abstraction of a Transport Plugin network interface.

A Transport Plugin may be able to use several logical or physical network interfaces in a single node (machine). For example, there may be multiple NICs for IP networks or multiple serial ports for serial networks.

An instance of a Transport Plugin is a conduit to one or more network interfaces associated with the transport.

Instances of a Transport Plugin must assign a unique unicast address to each of the network interfaces that it can use to send and receive messages. The unicast address should be within the range of addresses that are addressable by the Transport Plugin (as defined by the plugin itself).

Then, when RTI Connext sends a message to an unicast destination address, the destination address will be made of two parts. The network address and an interface address. The network address portion will be used by RTI Connext to select the Transport Plugin instance that will send the message. The interface address portion is passed to the Transport Plugin instance as the destination interface to which the message should be sent.

See also

NDDS_Transport_Address_t (p. 2042) **NDDS_Transport_ClassId_t** (p. 447)

7.110.2 Enumeration Type Documentation

7.110.2.1 NDDS_Transport_Interface_Status_t

`enum` **`NDDS_Transport_Interface_Status_t`**

Interface status.

Enumerator

NDDS_TRANSPORT_INTERFACE_OFF	The transport interface is OFF.
NDDS_TRANSPORT_INTERFACE_ON	The transport interface is ON.

7.111 Transport Plugins Configuration

Transport plugins configuration with RTI Connext.

Classes

- struct **NDDS_Transport_UUID**

  Unequivocally identifies a transport plugin instance.
- struct **TransportAllocationSettings_t**

  Allocation settings used by various internal buffers.
- struct **NDDS_Transport_Property_t**

  Base configuration structure that must be inherited by derived Transport Plugin classes.

Macros

- #define **NDDS_TRANSPORT_PORT_INVALID** ((**NDDS_Transport_Port_t**) 0)

  Port 0 is considered to be invalid.
- #define **NDDS_TRANSPORT_UUID_SIZE** 12

  *Size of a **NDDS_Transport_UUID** (p. 2087).*
- #define **NDDS_TRANSPORT_LENGTH_UNLIMITED** -1

  Represent an unlimited length.
- #define **NDDS_TRANSPORT_INTERFACE_RANK_UNKNOWN** 0

  Rank interface as unknown or not yet set.
- #define **NDDS_TRANSPORT_UUID_UNKNOWN**

  Value for UUIDs that have no known value. Used as default.
- #define **NDDS_TRANSPORT_ALLOCATION_SETTINGS_MAX_COUNT_UNLIMITED** (-1)

  *The constant used as 'unlimited' for the 'max_count' field of the structure **TransportAllocationSettings_t** (p. 2218).*
- #define **NDDS_TRANSPORT_ALLOCATION_SETTINGS_INCREMENTAL_COUNT_AUTOMATIC** (-1)

  *The constant used as 'automatic' for the 'incremental_count' field of the structure **TransportAllocationSettings_t** (p. 2218).*
- #define **NDDS_TRANSPORT_ALLOCATION_SETTINGS_DEFAULT**

  *The constant used as default value for the struct **TransportAllocationSettings_t** (p. 2218).*
- #define **NDDS_TRANSPORT_CLASSID_INVALID** (-1)

  Invalid Transport Class ID.
- #define **NDDS_TRANSPORT_CLASSID_UDPv4** (1)

  Builtin IPv4 UDP/IP Transport Plugin class ID.
- #define **NDDS_TRANSPORT_CLASSID_SHMEM** (0x01000000)

  Builtin Shared-Memory Transport Plugin class ID.

- #define **NDDS_TRANSPORT_CLASSID_SHMEM_510** (2)

  Builtin Shared-Memory Transport Plugin class ID for Connext 5.1.0 and earlier.
- #define **NDDS_TRANSPORT_CLASSID_UDPv6** (2)

  Builtin IPv6 UDP/IP Transport Plugin class ID.
- #define **NDDS_TRANSPORT_CLASSID_UDPv6_510** (5)

  Builtin IPv6 UDP/IP Transport Plugin class ID for Connext 5.1.0 and earlier.
- #define **NDDS_TRANSPORT_CLASSID_TCPV4_LAN** (8)

  IPv4 TCP/IP Transport Plugin class ID for LAN case.
- #define **NDDS_TRANSPORT_CLASSID_TCPV4_WAN** (9)

  IPv4 TCP/IP Transport Plugin class ID for WAN case.
- #define **NDDS_TRANSPORT_CLASSNAME_TCPV4_WAN** "tcpv4_wan"

  IPv4 TCP/IP Transport Plugin class name for WAN case.
- #define **NDDS_TRANSPORT_CLASSID_TLSV4_LAN** (10)

  IPv4 TCP/IP Transport Plugin class ID for LAN case with TLS enabled.
- #define **NDDS_TRANSPORT_CLASSID_TLSV4_WAN** (11)

  IPv4 TCP/IP Transport Plugin class ID for WAN case with TLS enabled.
- #define **NDDS_TRANSPORT_CLASSID_UDPv4_WAN** (0x01000001)

  Builtin IPv4 UDP/IP Asymmetric Transport Plugin class ID.
- #define **NDDS_TRANSPORT_CLASSID_RESERVED_RANGE** (1000)

  Transport Plugin class IDs below this are reserved by RTI.
- #define **NDDS_TRANSPORT_PROPERTY_BIT_BUFFER_ALWAYS_LOANED** (0x2)

  Specified zero-copy behavior of transport.
- #define **NDDS_TRANSPORT_PROPERTY_GATHER_SEND_BUFFER_COUNT_MIN** (3)

  Minimum number of gather-send buffers that must be supported by a Transport Plugin implementation.

Typedefs

- typedef RTI_UINT32 **NDDS_Transport_Port_t**

  Type for storing RTI Connext RTPS ports.
- typedef RTI_INT32 **NDDS_Transport_ClassId_t**

  Type for storing RTI Connext Transport Plugin class IDs.

7.111.1 Detailed Description

Transport plugins configuration with RTI Connext.

Transport plugins are configured using properties. Each transport plugin must derive its property from a base configuration structure **NDDS_Transport_Property_t** (p. 2044).

To see how to configure the Built-in Transport Plugins, see **Built-in Transport Plugins** (p. 182).

See also

Built-in Transport Plugins (p. 182)

7.111.2 Macro Definition Documentation

7.111.2.1 NDDS_TRANSPORT_PORT_INVALID

```
#define NDDS_TRANSPORT_PORT_INVALID (( NDDS_Transport_Port_t) 0)
```

Port 0 is considered to be invalid.

7.111.2.2 NDDS_TRANSPORT_UUID_SIZE

```
#define NDDS_TRANSPORT_UUID_SIZE 12
```

Size of a **NDDS_Transport_UUID** (p. 2087).

7.111.2.3 NDDS_TRANSPORT_LENGTH_UNLIMITED

```
#define NDDS_TRANSPORT_LENGTH_UNLIMITED -1
```

Represent an unlimited length.

7.111.2.4 NDDS_TRANSPORT_INTERFACE_RANK_UNKNOWN

```
#define NDDS_TRANSPORT_INTERFACE_RANK_UNKNOWN 0
```

Rank interface as unknown or not yet set.

7.111.2.5 NDDS_TRANSPORT_UUID_UNKNOWN

```
#define NDDS_TRANSPORT_UUID_UNKNOWN
```

Value:

```
{ \
  { \
    0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0 \
  } \
}
```

Value for UUIDs that have no known value. Used as default.

7.111.2.6 NDDS_TRANSPORT_ALLOCATION_SETTINGS_MAX_COUNT_UNLIMITED

```
#define NDDS_TRANSPORT_ALLOCATION_SETTINGS_MAX_COUNT_UNLIMITED (-1)
```

The constant used as 'unlimited' for the 'max_count' field of the structure **TransportAllocationSettings_t** (p. 2218).

7.111.2.7 NDDS_TRANSPORT_ALLOCATION_SETTINGS_INCREMENTAL_COUNT_AUTOMATIC

```
#define NDDS_TRANSPORT_ALLOCATION_SETTINGS_INCREMENTAL_COUNT_AUTOMATIC (-1)
```

The constant used as 'automatic' for the 'incremental_count' field of the structure **TransportAllocationSettings_t** (p. 2218).

Automatic means the buffer size will double at every reallocation.

7.111.2.8 NDDS_TRANSPORT_ALLOCATION_SETTINGS_DEFAULT

```
#define NDDS_TRANSPORT_ALLOCATION_SETTINGS_DEFAULT
```

Value:

```
        {                                   \
        2L, /* initial_count */                                                 \
        NDDS_TRANSPORT_ALLOCATION_SETTINGS_MAX_COUNT_UNLIMITED, /* max_count */     \
        NDDS_TRANSPORT_ALLOCATION_SETTINGS_INCREMENTAL_COUNT_AUTOMATIC /* incremental_count */ \
}
```

The constant used as default value for the struct **TransportAllocationSettings_t** (p. 2218).

The default value defined in this constant, sets the buffer to have:

- initial_count = 2 elements
- max_count = **NDDS_TRANSPORT_ALLOCATION_SETTINGS_MAX_COUNT_UNLIMITED** (p. 443)
- incremental_count = NDDS_TRANSPORT_ALLOCATION_SETTINGS_INCREMENTAL_COUNT_AUTOMATIC

7.111.2.9 NDDS_TRANSPORT_CLASSID_INVALID

```
#define NDDS_TRANSPORT_CLASSID_INVALID (-1)
```

Invalid Transport Class ID.

Transport-Plugins implementations should set their class ID to a value different than this.

7.111.2.10 NDDS_TRANSPORT_CLASSID_UDPv4

```
#define NDDS_TRANSPORT_CLASSID_UDPv4 (1)
```

Builtin IPv4 UDP/IP Transport Plugin class ID.

7.111.2.11 NDDS_TRANSPORT_CLASSID_SHMEM

```
#define NDDS_TRANSPORT_CLASSID_SHMEM (0x01000000)
```

Builtin Shared-Memory Transport Plugin class ID.

7.111.2.12 NDDS_TRANSPORT_CLASSID_SHMEM_510

```
#define NDDS_TRANSPORT_CLASSID_SHMEM_510 (2)
```

Builtin Shared-Memory Transport Plugin class ID for Connext 5.1.0 and earlier.

7.111.2.13 NDDS_TRANSPORT_CLASSID_UDPv6

```
#define NDDS_TRANSPORT_CLASSID_UDPv6 (2)
```

Builtin IPv6 UDP/IP Transport Plugin class ID.

7.111.2.14 NDDS_TRANSPORT_CLASSID_UDPv6_510

```
#define NDDS_TRANSPORT_CLASSID_UDPv6_510 (5)
```

Builtin IPv6 UDP/IP Transport Plugin class ID for Connext 5.1.0 and earlier.

7.111.2.15 NDDS_TRANSPORT_CLASSID_TCPV4_LAN

```
#define NDDS_TRANSPORT_CLASSID_TCPV4_LAN (8)
```

IPv4 TCP/IP Transport Plugin class ID for LAN case.

7.111.2.16 NDDS_TRANSPORT_CLASSID_TCPV4_WAN

```
#define NDDS_TRANSPORT_CLASSID_TCPV4_WAN (9)
```

IPv4 TCP/IP Transport Plugin class ID for WAN case.

7.111.2.17 NDDS_TRANSPORT_CLASSNAME_TCPV4_WAN

```
#define NDDS_TRANSPORT_CLASSNAME_TCPV4_WAN "tcpv4_wan"
```

IPv4 TCP/IP Transport Plugin class name for WAN case.

7.111.2.18 NDDS_TRANSPORT_CLASSID_TLSV4_LAN

```
#define NDDS_TRANSPORT_CLASSID_TLSV4_LAN (10)
```

IPv4 TCP/IP Transport Plugin class ID for LAN case with TLS enabled.

7.111.2.19 NDDS_TRANSPORT_CLASSID_TLSV4_WAN

```
#define NDDS_TRANSPORT_CLASSID_TLSV4_WAN (11)
```

IPv4 TCP/IP Transport Plugin class ID for WAN case with TLS enabled.

7.111.2.20 NDDS_TRANSPORT_CLASSID_UDPv4_WAN

```
#define NDDS_TRANSPORT_CLASSID_UDPv4_WAN (0x01000001)
```

Builtin IPv4 UDP/IP Asymmetric Transport Plugin class ID.

7.111.2.21 NDDS_TRANSPORT_CLASSID_RESERVED_RANGE

```
#define NDDS_TRANSPORT_CLASSID_RESERVED_RANGE (1000)
```

Transport Plugin class IDs below this are reserved by RTI.

User-defined Transport-Plugins should use a class ID greater than this number.

7.111.2.22 NDDS_TRANSPORT_PROPERTY_BIT_BUFFER_ALWAYS_LOANED

```
#define NDDS_TRANSPORT_PROPERTY_BIT_BUFFER_ALWAYS_LOANED (0x2)
```

Specified zero-copy behavior of transport.

A Transport Plugin may commit to one of three behaviors for zero copy receives:

1. Always does zero copy.
2. Sometimes does zero copy, up to the transport discretion.
3. Never does zero copy.

This bit should be set only if the Transport Plugin commits to always doing a zero copy receive, or more specifically, always loaning a buffer through its `receive_rEA()` call.

In that case, RTI Connext will not need to allocate storage for a message that it retrieves with the `receive_rEA()` call.

7.111.2.23 NDDS_TRANSPORT_PROPERTY_GATHER_SEND_BUFFER_COUNT_MIN

```
#define NDDS_TRANSPORT_PROPERTY_GATHER_SEND_BUFFER_COUNT_MIN (3)
```

Minimum number of gather-send buffers that must be supported by a Transport Plugin implementation.

For the **NDDS_Transport_Property_t** (p. 2044) structure to be valid, the value of **NDDS_Transport_Property_t↩ ::gather_send_buffer_count_max** (p. 2047) must be greater than or equal to this value.

7.111.3 Typedef Documentation

7.111.3.1 NDDS_Transport_Port_t

```
typedef RTI_UINT32 NDDS_Transport_Port_t
```

Type for storing RTI Connext RTPS ports.

Unlike IPv4 Socket API ports, which are 2 bytes long, the RTI Connext representation of an RTPS port is 4 bytes.

7.111.3.2 NDDS_Transport_ClassId_t

```
typedef RTI_INT32 NDDS_Transport_ClassId_t
```

Type for storing RTI Connext Transport Plugin class IDs.

Each implementation of a Transport Plugin must have a unique ID. For example, a UDP/IP Transport Plugin implementation must have a different ID than a Shared Memory Transport Plugin.

User-implemented Transport Plugins must have an ID higher than **NDDS_TRANSPORT_CLASSID_RESERVED_RANGE** (p. 446).

7.112 Transport Address

Transport-independent addressing scheme using IPv6 presentation strings and numerically stored in network-ordered format.

Classes

- struct **NDDS_Transport_Address_t**

  Addresses are stored individually as network-ordered bytes.

Macros

- #define **NDDS_TRANSPORT_ADDRESS_INVALID_INITIALIZER**

  An invalid transport address. Used as an initializer.
- #define **NDDS_TRANSPORT_ADDRESS_STRING_BUFFER_SIZE** (72)

  *The minimum size of the buffer that should be passed to **NDDS_Transport_Address_to_string** (p. 450).*

Functions

- RTIBool **NDDS_Transport_Address_to_string** (const **NDDS_Transport_Address_t** *self, char *buffer_inout, size_t buffer_size_in)

  Converts a numerical address to a printable string representation.
- RTIBool **NDDS_Transport_Address_to_string_with_protocol_family_format** (const **NDDS_Transport_Address_t** *me, char *buffer, size_t buffer_size_in, RTIOsapiSocketAFKind family)

  Converts a numerical address to a printable string representation with IPv4 dotted notation or IPv6 presentation string depending on the provided protocol family.
- RTI_INT32 **NDDS_Transport_Address_from_string** (**NDDS_Transport_Address_t** *address_out, const char *address_in)

  Converts an address (IPv4 dotted notation or IPv6 presentation string) into a numerical address.
- void **NDDS_Transport_Address_print** (const **NDDS_Transport_Address_t** *address_in, const char *desc_in, RTI_INT32 indent_in)

  Prints an address to standard out.
- RTI_INT32 **NDDS_Transport_Address_is_ipv4** (const **NDDS_Transport_Address_t** *address_in)

  Checks if an address is an IPv4 address.
- RTI_INT32 **NDDS_Transport_Address_is_multicast** (const **NDDS_Transport_Address_t** *address_in)

  Checks if an address is an IPv4 or IPv6 multicast address.

Variables

- const **NDDS_Transport_Address_t NDDS_TRANSPORT_ADDRESS_INVALID**

  An invalid transport address.

7.112.1 Detailed Description

Transport-independent addressing scheme using IPv6 presentation strings and numerically stored in network-ordered format.

The APIs of RTI Connext uses IPv6 address notation for all transports.

Transport Plugin implementations that are not IP-based are required to map whatever addressing scheme natively used by the physical transport (if any) to an address in IPv6 notation and vice versa.

IPv6 addresses are numerically stored in 16 bytes. An IPv6 address can be presented In string notation in a variety of ways. For example,

```
"00AF:0000:0037:FE01:0000:0000:034B:0089"
"AF:0:37:FE01:0:0:34B:89"
"AF:0:37:FE01::34B:89"
```

are all valid IPv6 presentation of the same address.

IPv4 address in dot notation can be used to specify the last 4 bytes of the address. For example,

```
"0000:0000:0000:0000:0000:0000:192.168.0.1"
"0:0:0:0:0:0:192.168.0.1"
"::192.168.0.1"
```

are all valid IPv6 presentation of the same address.

For a complete description of valid IPv6 address notation, consult the IPv6 Addressing Architecture (RFC 2373).

Addresses are divided into unicast addresses and multicast addresses.

Multicast addresses are defined as

- Addresses that start with 0xFF. That is `FFxx:xxxx:xxxx:xxxx:xxxx:xxxx:xxxx:xxxx`.

or an IPv4 multicast address

- Address in the range [::224.0.0.0, ::239.255.255.255]

Multicast addresses do not refer to any specific destination (network interface). Instead, they usually refer to a group of network interfaces, often called a "multicast group".

Unicast addresses always refer to a specific network interface.

7.112.2 Macro Definition Documentation

7.112.2.1 NDDS_TRANSPORT_ADDRESS_INVALID_INITIALIZER

```
#define NDDS_TRANSPORT_ADDRESS_INVALID_INITIALIZER
```

Value:

```
  { \
    { \
      0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0 \
    } \
  }
```

An invalid transport address. Used as an initializer.

For example: **NDDS_Transport_Address_t** (p. 2042) address = NDDS_TRANSPORT_ADDRESS_INVALID_↩ INITIALIZER;

7.112.2.2 NDDS_TRANSPORT_ADDRESS_STRING_BUFFER_SIZE

```
#define NDDS_TRANSPORT_ADDRESS_STRING_BUFFER_SIZE (72)
```

The minimum size of the buffer that should be passed to **NDDS_Transport_Address_to_string** (p. 450).

For regular addresses, the string size needs to be at least 40 to include space for 8 tuples of 4 characters each plus 7 delimiting colons plus a terminating NULL.

To support UDPv4_WAN strings, it has been adjusted to 72 to fit the following representation (plus NULL terminator): f=XXXXRBPU,u={FF,FF,FF,FF,FF,FF,FF,FF},p=255.255.255.255:65555:65555

7.112.3 Function Documentation

7.112.3.1 NDDS_Transport_Address_to_string()

```
RTIBool NDDS_Transport_Address_to_string (
            const  NDDS_Transport_Address_t * self,
            char * buffer_inout,
            size_t buffer_size_in )
```

Converts a numerical address to a printable string representation.

Precondition

The `buffer_inout` provided must be at least **NDDS_TRANSPORT_ADDRESS_STRING_BUFFER_SIZE** (p. 450) characters long.

Parameters

self	<<***in***>> *(p. 433)* The address to be converted.
buffer_inout	<<***inout***>> *(p. 433)* Storage passed in which to return the string corresponding to the address.
buffer_size_in	<<***in***>> *(p. 433)* The size of the storage buffer. Must be >= **NDDS_TRANSPORT_ADDRESS_STRING_BUFFER_SIZE** (p. 450)

Returns

1 upon success; 0 upon failure (not enough space in the provided buffer)

7.112.3.2 NDDS_Transport_Address_to_string_with_protocol_family_format()

```
RTIBool NDDS_Transport_Address_to_string_with_protocol_family_format (
            const  NDDS_Transport_Address_t * me,
            char * buffer,
            size_t buffer_size_in,
            RTIOsapiSocketAFKind family )
```

Converts a numerical address to a printable string representation with IPv4 dotted notation or IPv6 presentation string depending on the provided protocol family.

Precondition

The `buffer_inout` provided must be at least **NDDS_TRANSPORT_ADDRESS_STRING_BUFFER_SIZE** (p. 450) characters long.

Parameters

me	<<***in***>> *(p. 433)* The address to be converted.
buffer	<<***inout***>> *(p. 433)* Storage passed in which to return the string corresponding to the address.
buffer_size_in	<<***in***>> *(p. 433)* The size of the storage buffer. Must be >= **NDDS_TRANSPORT_ADDRESS_STRING_BUFFER_SIZE** (p. 450)
family	<<***in***>> *(p. 433)* The protocol family RTI_OSAPI_SOCKET_AF_INET or RTI_OSAPI_SOCKET_AF_INET6

Returns

RTI_TRUE upon success; 0 upon failure (not enough space in the provided buffer)

7.112.3.3 NDDS_Transport_Address_from_string()

```
RTI_INT32 NDDS_Transport_Address_from_string (
              NDDS_Transport_Address_t * address_out,
             const char * address_in )
```

Converts an address (IPv4 dotted notation or IPv6 presentation string) into a numerical address.

The address string must be in IPv4 dotted notation (X.X.X.X) or IPv6 presentation notation. The string cannot be a hostname since this function does not perform a hostname lookup.

Parameters

address_out	<<***out***>> *(*p. *433)* Numerical value of the address.
address_in	<<***in***>> *(*p. *433)* String representation of an address.

Returns

1 if address_out contains a valid address, 0 if it was not able to convert the string into an address.

7.112.3.4 NDDS_Transport_Address_print()

```
void NDDS_Transport_Address_print (
             const  NDDS_Transport_Address_t * address_in,
             const char * desc_in,
             RTI_INT32 indent_in )
```

Prints an address to standard out.

Parameters

address_in	<<***in***>> *(*p. *433)* Address to be printed.
desc_in	<<***in***>> *(*p. *433)* A prefix to be printed before the address.
indent_in	<<***in***>> *(*p. *433)* Indentation level for the printout.

7.112.3.5 NDDS_Transport_Address_is_ipv4()

```
RTI_INT32 NDDS_Transport_Address_is_ipv4 (
             const  NDDS_Transport_Address_t * address_in )
```

Checks if an address is an IPv4 address.

Parameters

address_in	<<**in**>> *(p. 433)* Address to be tested.

Note

May be implemented as a macro for efficiency.

Returns

1 if address is an IPv4 address, 0 otherwise.

7.112.3.6 NDDS_Transport_Address_is_multicast()

```
RTI_INT32 NDDS_Transport_Address_is_multicast (
            const  NDDS_Transport_Address_t * address_in )
```

Checks if an address is an IPv4 or IPv6 multicast address.

Parameters

address_in	<<**in**>> *(p. 433)* Address to be tested.

May be implemented as a macro for efficiency.

Returns

1 if address is a multicast address, 0 otherwise.

7.112.4 Variable Documentation

7.112.4.1 NDDS_TRANSPORT_ADDRESS_INVALID

```
const  NDDS_Transport_Address_t NDDS_TRANSPORT_ADDRESS_INVALID  [extern]
```

An invalid transport address.

7.113 UDP Transport Plugin definitions

UDP Transport Plugin definitions.

Classes

- struct **NDDS_Transport_UDP_WAN_CommPortsMappingInfo**

  Type for storing UDP WAN communication ports.

Macros

- #define **NDDS_TRANSPORT_UDP_PROPERTIES_BITMAP_DEFAULT** (0)

  *Default value of **NDDS_Transport_Property_t::properties_bitmap*** (p. *2046*).
- #define **NDDS_TRANSPORT_UDP_GATHER_SEND_BUFFER_COUNT_MAX_DEFAULT** (16)

  *Default value of **NDDS_Transport_Property_t::gather_send_buffer_count_max*** (p. *2047*).
- #define **NDDS_TRANSPORT_UDP_SOCKET_BUFFER_SIZE_OS_MAX** RTI_OSAPI_UDP_SOCKET_↩ BUFFER_SIZE_OS_MAX

  Used to specify that os maximum be used to specify socket buffer size.
- #define **NDDS_TRANSPORT_UDP_SOCKET_BUFFER_SIZE_OS_DEFAULT** RTI_OSAPI_UDP_SOCKET↩ _BUFFER_SIZE_OS_DEFAULT

  Used to specify that os default be used to specify socket buffer size.
- #define **NDDS_TRANSPORT_UDP_SEND_SOCKET_BUFFER_SIZE_DEFAULT** RTI_OSAPI_UDP_SEND↩ _SOCKET_BUFFER_SIZE_DEFAULT

  Default value of send_socket_buffer_size.
- #define **NDDS_TRANSPORT_UDP_RECV_SOCKET_BUFFER_SIZE_DEFAULT** RTI_OSAPI_UDP_RECV↩ _SOCKET_BUFFER_SIZE_DEFAULT

  Default value of recv_socket_buffer_size.
- #define **NDDS_TRANSPORT_UDP_MULTICAST_TTL_DEFAULT** (1)

  Default value of multicast_ttl.

Typedefs

- typedef RTI_UINT16 **NDDS_Transport_UDP_Port**

  UDP port.

7.113.1 Detailed Description

UDP Transport Plugin definitions.

7.113.2 Macro Definition Documentation

7.113.2.1 NDDS_TRANSPORT_UDP_PROPERTIES_BITMAP_DEFAULT

```
#define NDDS_TRANSPORT_UDP_PROPERTIES_BITMAP_DEFAULT (0)
```

Default value of **NDDS_Transport_Property_t::properties_bitmap** (p. 2046).

7.113.2.2 NDDS_TRANSPORT_UDP_GATHER_SEND_BUFFER_COUNT_MAX_DEFAULT

```
#define NDDS_TRANSPORT_UDP_GATHER_SEND_BUFFER_COUNT_MAX_DEFAULT (16)
```

Default value of **NDDS_Transport_Property_t::gather_send_buffer_count_max** (p. 2047).

This is also the maximum value that can be used when instantiating the udp transport.

16 is sufficient for RTI Connext, but more may improve discovery and reliable performance. Porting note: find out what the maximum gather buffer count is on your OS!

7.113.2.3 NDDS_TRANSPORT_UDP_SOCKET_BUFFER_SIZE_OS_MAX

```
#define NDDS_TRANSPORT_UDP_SOCKET_BUFFER_SIZE_OS_MAX  RTI_OSAPI_UDP_SOCKET_BUFFER_SIZE_OS_MAX
```

Used to specify that os maximum be used to specify socket buffer size.

This property, if set, uses the maximum allowable size set in your OS for send and receive socket buffer sizes. If you update the maximum allowable size in your OS setting, then RTI Connext will automatically pick up that change. This property is not supported on all platforms because not all platforms have maximum values for the size of the send and receive buffers for sockets. If the platform does not support this property, then the property will be set to **NDDS_Transport_UDP_Plugin::NDDS_TRANSPORT_UDP_SEND_SOCKET_BUFFER_SIZE_DEFAULT** (p. 455) or **NDDS_Transport_UDP_Plugin::NDDS_TRANSPORT_UDP_RECV_SOCKET_BUFFER_SIZE_DEFAULT** (p. 456), depending on the property being set. See the Platform Notes to determine if this property is supported on your platform.

This value is -2.

7.113.2.4 NDDS_TRANSPORT_UDP_SOCKET_BUFFER_SIZE_OS_DEFAULT

```
#define NDDS_TRANSPORT_UDP_SOCKET_BUFFER_SIZE_OS_DEFAULT  RTI_OSAPI_UDP_SOCKET_BUFFER_SIZE_OS_↩
DEFAULT
```

Used to specify that os default be used to specify socket buffer size.

This property, if set, uses the default size set in your OS for send and receive socket buffer sizes. If you update the default size in your OS settings, then RTI Connext will automatically pick up that change.

This value is -1.

7.113.2.5 NDDS_TRANSPORT_UDP_SEND_SOCKET_BUFFER_SIZE_DEFAULT

```
#define NDDS_TRANSPORT_UDP_SEND_SOCKET_BUFFER_SIZE_DEFAULT  RTI_OSAPI_UDP_SEND_SOCKET_BUFFER_↩
SIZE_DEFAULT
```

Default value of send_socket_buffer_size.

The default value is 2097152 bytes (1048576 bytes on AIX platforms).

7.113.2.6 NDDS_TRANSPORT_UDP_RECV_SOCKET_BUFFER_SIZE_DEFAULT

```
#define NDDS_TRANSPORT_UDP_RECV_SOCKET_BUFFER_SIZE_DEFAULT  RTI_OSAPI_UDP_RECV_SOCKET_BUFFER_↩
SIZE_DEFAULT
```

Default value of recv_socket_buffer_size.

The default value is 2097152 bytes (1048576 bytes on AIX platforms).

7.113.2.7 NDDS_TRANSPORT_UDP_MULTICAST_TTL_DEFAULT

```
#define NDDS_TRANSPORT_UDP_MULTICAST_TTL_DEFAULT (1)
```

Default value of multicast_ttl.

7.113.3 Typedef Documentation

7.113.3.1 NDDS_Transport_UDP_Port

```
typedef RTI_UINT16  NDDS_Transport_UDP_Port
```

UDP port.

7.114 Shared Memory Transport

Built-in transport plug-in for inter-process communications using shared memory (**NDDS_TRANSPORT_CLASSID_↩SHMEM** (p. 445)) .

Classes

- struct **NDDS_Transport_Shmem_Property_t**

  *Subclass of **NDDS_Transport_Property_t** (p. 2044) allowing specification of parameters that are specific to the shared-memory transport.*

Macros

- #define **NDDS_TRANSPORT_SHMEM_ADDRESS_BIT_COUNT** (-96)

  *Default value of **NDDS_Transport_Property_t::address_bit_count** (p. 2046).*
- #define **NDDS_TRANSPORT_SHMEM_PROPERTIES_BITMAP_DEFAULT** (**NDDS_TRANSPORT_↩ PROPERTY_BIT_BUFFER_ALWAYS_LOANED**)

  *Default value of **NDDS_Transport_Property_t::properties_bitmap** (p. 2046).*
- #define **NDDS_TRANSPORT_SHMEM_GATHER_SEND_BUFFER_COUNT_MAX_DEFAULT** (1024)

  *Default value of **NDDS_Transport_Property_t::gather_send_buffer_count_max** (p. 2047).*
- #define **NDDS_TRANSPORT_SHMEM_MESSAGE_SIZE_MAX_DEFAULT** (65536)

  *Default value of **NDDS_Transport_Property_t::message_size_max** (p. 2047).*
- #define **NDDS_TRANSPORT_SHMEM_RECEIVED_MESSAGE_COUNT_MAX_DEFAULT** (64)

  *Default value of **NDDS_Transport_Shmem_Property_t::received_message_count_max** (p. 2053).*
- #define **NDDS_TRANSPORT_SHMEM_RECEIVE_BUFFER_SIZE_DEFAULT**

  *Default value of **NDDS_Transport_Shmem_Property_t::receive_buffer_size** (p. 2053).*
- #define **NDDS_TRANSPORT_SHMEM_MAJOR_AFTER_BUG_14240_FIX** (2)

  Major version for the transport plugin after fixing bug 14240 (RTI-28)
- #define **NDDS_TRANSPORT_SHMEM_PROPERTY_DEFAULT**

  Use this to initialize stack variable.

Functions

- NDDS_Transport_Plugin * **NDDS_Transport_Shmem_new** (const struct **NDDS_Transport_Shmem_↩ Property_t** *property_in)

  Create a new shmem process transport.
- NDDS_Transport_Plugin * **NDDS_Transport_Shmem_create** (**NDDS_Transport_Address_t** *default_↩ network_address_out, const struct **DDS_PropertyQosPolicy** *property_in, const struct **DDS_Product↩ Version_t** *minimum_compatibility_version)

  Create a new shmem process transport, using PropertyQosPolicy.

7.114.1 Detailed Description

Built-in transport plug-in for inter-process communications using shared memory (**NDDS_TRANSPORT_CLASSID_↩ SHMEM** (p. 445)) .

This transport plugin uses System Shared Memory to send messages between processes on the same node. This transport is installed as a built-in transport plugin with the alias **DDS_TRANSPORTBUILTIN_SHMEM_ALIAS** (p. 644).

7.114.2 Compatibility of Sender and Receiver Transports

Opening a receiver "port" on shared memory corresponds to creating a shared memory segment using a name based on the port number. The transport plugin's properties are embedded in the shared memory segment.

When a sender tries to send to the shared memory port, it verifies that properties of the receiver's shared memory transport are compatible with those specified in its transport plugin. If not, the sender will fail to attach to the port and will output messages such as below (with numbers appropriate to the properties of the transport plugins involved).

```
NDDS_Transport_Shmem_attachShmem:failed to initialize incompatible properties

NDDS_Transport_Shmem_attachShmem:countMax 0 > -19417345 or max size -19416188 > 2147482624
```

In this scenario, the properties of the sender or receiver transport plugin instances should be adjusted, so that they are compatible.

7.114.3 Crashing and Restarting Programs

If a process using shared memory crashes (say because the user typed in ^C), resources associated with its shared memory ports may not be properly cleaned up. Later, if another RTI Connext process needs to open the same ports (say, the crashed program is restarted), it will attempt to reuse the shared memory segment left behind by the crashed process.

The reuse is allowed iff the properties of transport plugin are compatible with those embedded in the shared memory segment (i.e., of the original creator). Otherwise, the process will fail to open the ports, and will output messages such as below (with numbers appropriate to the properties of the transport plugins involved).

```
NDDS_Transport_Shmem_create_recvresource_rrEA:failed to initialize shared

    memory resource Cannot recycle existing shmem: size not compatible for key 0x1234
```

In this scenario, the shared memory segments must be cleaned up using appropriate platform specific commands. For details, please refer to the `Platform Notes`.

7.114.4 Shared Resource Keys

The transport uses the **shared memory segment keys**, given by the formula below.
`0x400000 + port`

The transport also uses signaling **shared semaphore** keys given by the formula below.
`0x800000 + port`

The transport also uses mutex **shared semaphore keys** given by the formula below.
`0xb00000 + port`

wher the `port` is a function of the `domain_id` and the `participant_id`, as described in **DDS_WireProtocol↩ QosPolicy::participant_id** (p. 1514)

See also

DDS_WireProtocolQosPolicy::participant_id (p. 1514)

NDDSTransportSupport::set_builtin_transport_property() (p. 2109)

7.114.5 Creating and Registering Shared Memory Transport Plugin

RTI Connext can implicitly create this plugin and register with the **DDSDomainParticipant** (p. 1619) if this transport is specified in **DDS_TransportBuiltinQosPolicy** (p. 1408).

To specify the properties of the builtin shared memory transport that is implicitly registered, you can either:

- call **NDDSTransportSupport::set_builtin_transport_property** (p. 2109) or
- specify the pre-defined property names in **DDS_PropertyQosPolicy** (p. 1260) associated with the **DDSDomain↩ Participant** (p. 1619). (see **Shared Memory Transport Property Names in Property QoS Policy of Domain Participant** (p. 459)).

Builtin transport plugin properties specified in **DDS_PropertyQosPolicy** (p. 1260) always overwrite the ones specified through **NDDSTransportSupport::set_builtin_transport_property()** (p. 2109). The default value is assumed on any unspecified property. Note that all properties should be set before the transport is implicitly created and registered by RTI Connext. See **Built-in Transport Plugins** (p. 182) for details on when a builtin transport is registered.

To explicitly create an instance of this plugin, **NDDS_Transport_Shmem_new()** (p. 461) should be called. The instance should be registered with RTI Connext, see **NDDSTransportSupport::register_transport** (p. 2104). In some configurations, you may have to disable the builtin shared memory transport plugin instance (**DDS_TransportBuiltinQosPolicy** (p. 1408), **DDS_TRANSPORTBUILTIN_SHMEM** (p. 644)), to avoid port conflicts with the newly created plugin instance.

7.114.6 Shared Memory Transport Property Names in Property QoS Policy of Domain Participant

The following table lists the predefined property names that can be set in the **DDS_DomainParticipantQos::property** (p. 975) to to configure the builtin shared memory transport plugin.

Table 7.152 ***Property Strings for Shared Memory Transport***

Name	Descriptions
dds.transport.shmem.builtin.parent.address_bit_count	See **NDDS_Transport_Property_t::address_bit_count** (p. 2046)
dds.transport.shmem.builtin.parent.properties_bitmap	See **NDDS_Transport_Property_t::properties_bitmap** (p. 2046)
dds.transport.shmem.builtin.parent.gather_send_buffer_count_max	See **NDDS_Transport_Property_t::gather_send_buffer_count_max** (p. 2047)
dds.transport.shmem.builtin.parent.message_size_max	See **NDDS_Transport_Property_t::message_size_max** (p. 2047)
dds.transport.shmem.builtin.parent.allow_interfaces	See **NDDS_Transport_Property_t::allow_interfaces_list** (p. 2047) and **NDDS_Transport_Property_t::allow_interfaces_list_length** (p. 2048). Interfaces should be specified as comma-separated strings, with each comma delimiting an interface. For example: 127.0.0.1,eth0
dds.transport.shmem.builtin.parent.deny_interfaces	See **NDDS_Transport_Property_t::deny_interfaces_list** (p. 2048) and **NDDS_Transport_Property_t::deny_interfaces_list_length** (p. 2049). Interfaces should be specified as comma-separated strings, with each comma delimiting an interface. For example: 127.0.0.1,eth0
dds.transport.shmem.builtin.parent.allow_multicast_interfaces	See **NDDS_Transport_Property_t::allow_multicast_interfaces_list** (p. 2049) and **NDDS_Transport_Property_t::allow_multicast_interfaces_list_length** (p. 2050). Interfaces should be specified as comma-separated strings, with each comma delimiting an interface. For example: 127.0.0.1,eth0
dds.transport.shmem.builtin.parent.deny_multicast_interfaces	See **NDDS_Transport_Property_t::deny_multicast_interfaces_list** (p. 2050) and **NDDS_Transport_Property_t::deny_multicast_interfaces_list_length** (p. 2050). Interfaces should be specified as comma-separated strings, with each comma delimiting an interface. For example: 127.0.0.1,eth0
dds.transport.shmem.builtin.parent.max_interface_count	See **NDDS_Transport_Property_t::max_interface_count** (p. 2051)
dds.transport.shmem.builtin.parent.thread_name_prefix	See **NDDS_Transport_Property_t::thread_name_prefix** (p. 2051)
dds.transport.shmem.builtin.received_message_count_max	See **NDDS_Transport_Shmem_Property_t::received_message_count_max** (p. 2053)

Name	Descriptions
dds.transport.shmem.builtin.receive_buffer_size	See **NDDS_Transport_Shmem_Property_t::receive_buffer_size** (p. 2053)
dds.transport.shmem.builtin.enable_udp_debugging	See **NDDS_Transport_Shmem_Property_t::enable_udp_debugging** (p. 2054)
dds.transport.shmem.builtin.udp_debugging_address	See **NDDS_Transport_Shmem_Property_t::udp_debugging_address** (p. 2054)
dds.transport.shmem.builtin.udp_debugging_port	See **NDDS_Transport_Shmem_Property_t::udp_debugging_port** (p. 2055)

7.114.7 Macro Definition Documentation

7.114.7.1 NDDS_TRANSPORT_SHMEM_ADDRESS_BIT_COUNT

```
#define NDDS_TRANSPORT_SHMEM_ADDRESS_BIT_COUNT (-96)
```

Default value of **NDDS_Transport_Property_t::address_bit_count** (p. 2046).

7.114.7.2 NDDS_TRANSPORT_SHMEM_PROPERTIES_BITMAP_DEFAULT

```
#define NDDS_TRANSPORT_SHMEM_PROPERTIES_BITMAP_DEFAULT ( NDDS_TRANSPORT_PROPERTY_BIT_BUFFER_ALWAYS_LOANED)
```

Default value of **NDDS_Transport_Property_t::properties_bitmap** (p. 2046).

7.114.7.3 NDDS_TRANSPORT_SHMEM_GATHER_SEND_BUFFER_COUNT_MAX_DEFAULT

```
#define NDDS_TRANSPORT_SHMEM_GATHER_SEND_BUFFER_COUNT_MAX_DEFAULT (1024)
```

Default value of **NDDS_Transport_Property_t::gather_send_buffer_count_max** (p. 2047).

7.114.7.4 NDDS_TRANSPORT_SHMEM_MESSAGE_SIZE_MAX_DEFAULT

```
#define NDDS_TRANSPORT_SHMEM_MESSAGE_SIZE_MAX_DEFAULT (65536)
```

Default value of **NDDS_Transport_Property_t::message_size_max** (p. 2047).

7.114.7.5 NDDS_TRANSPORT_SHMEM_RECEIVED_MESSAGE_COUNT_MAX_DEFAULT

```
#define NDDS_TRANSPORT_SHMEM_RECEIVED_MESSAGE_COUNT_MAX_DEFAULT (64)
```

Default value of **NDDS_Transport_Shmem_Property_t::received_message_count_max** (p. 2053).

7.114.7.6 NDDS_TRANSPORT_SHMEM_RECEIVE_BUFFER_SIZE_DEFAULT

```
#define NDDS_TRANSPORT_SHMEM_RECEIVE_BUFFER_SIZE_DEFAULT
```

Value:

```
(NDDS_TRANSPORT_SHMEM_RECEIVED_MESSAGE_COUNT_MAX_DEFAULT \
 * NDDS_TRANSPORT_SHMEM_MESSAGE_SIZE_MAX_DEFAULT / 4)
```

Default value of **NDDS_Transport_Shmem_Property_t::receive_buffer_size** (p. 2053).

7.114.7.7 NDDS_TRANSPORT_SHMEM_MAJOR_AFTER_BUG_14240_FIX

```
#define NDDS_TRANSPORT_SHMEM_MAJOR_AFTER_BUG_14240_FIX (2)
```

Major version for the transport plugin after fixing bug 14240 (RTI-28)

7.114.7.8 NDDS_TRANSPORT_SHMEM_PROPERTY_DEFAULT

```
#define NDDS_TRANSPORT_SHMEM_PROPERTY_DEFAULT
```

Use this to initialize stack variable.

7.114.8 Function Documentation

7.114.8.1 NDDS_Transport_Shmem_new()

```
NDDS_Transport_Plugin * NDDS_Transport_Shmem_new (
            const struct  NDDS_Transport_Shmem_Property_t * property_in )
```

Create a new shmem process transport.

An application may create multiple transports, possibly for use in different domains.

Parameters

property_in	<<***in***>> *(p. 433)* Desired behavior of this transport. May be NULL for default property. The transport plugin can only support one unicast receive interface; therefore the interface selection lists are ignored.

Returns

handle to a Shmem inter-process Transport Plugin on success

NULL on failure.

7.114.8.2 NDDS_Transport_Shmem_create()

```
NDDS_Transport_Plugin * NDDS_Transport_Shmem_create (
             NDDS_Transport_Address_t * default_network_address_out,
            const struct  DDS_PropertyQosPolicy * property_in,
            const struct  DDS_ProductVersion_t * minimum_compatibility_version )
```

Create a new shmem process transport, using PropertyQosPolicy.

An application may create multiple transports, possibly for use in different domains.

Parameters

default_network_address_out	<<***out***>> *(p. 433)* Network address to be used when registering the transport.
property_in	<<***in***>> *(p. 433)* Desired behavior of this transport. May be NULL for default property. The transport plugin can only support one unicast receive interface; therefore the interface selection lists are ignored.
minimum_compatibility_version	<<***in***>> *(p. 433)* Desired product version for the shared memory transport. If NULL, the default shared memory transport is used (no additional compatibility restrictions for the transport version). Please refer to the minimum_compatibility_version property for more information.

Returns

handle to a Shmem inter-process Transport Plugin on success

NULL on failure.

7.115 UDPv4 Transport

Transport plug-in using UDP/IPv4 (**NDDS_TRANSPORT_CLASSID_UDPv4** (p. 444)) .

Classes

- struct **NDDS_Transport_UDPv4_Property_t**

  Configurable IPv4/UDP Transport-Plugin properties.

Macros

- #define **NDDS_TRANSPORT_UDPV4_ADDRESS_BIT_COUNT** (32)

  *Default value of **NDDS_Transport_Property_t::address_bit_count*** (p. *2046*).
- #define **NDDS_TRANSPORT_UDPV4_WAN_ADDRESS_BIT_COUNT** (128)

  *Default value of **NDDS_Transport_Property_t::address_bit_count*** (p. *2046*) *for UDPv4 asymmetric transport.*
- #define **NDDS_TRANSPORT_UDPV4_PROPERTIES_BITMAP_DEFAULT** **NDDS_TRANSPORT_UDP_↩ PROPERTIES_BITMAP_DEFAULT**

  *Default value of **NDDS_Transport_Property_t::properties_bitmap*** (p. *2046*).
- #define **NDDS_TRANSPORT_UDPV4_GATHER_SEND_BUFFER_COUNT_MAX_DEFAULT** **NDDS_↩ TRANSPORT_UDP_GATHER_SEND_BUFFER_COUNT_MAX_DEFAULT**

  *Default value of **NDDS_Transport_Property_t::gather_send_buffer_count_max*** (p. *2047*).
- #define **NDDS_TRANSPORT_UDPV4_SOCKET_BUFFER_SIZE_OS_MAX** **NDDS_TRANSPORT_UDP_↩ SOCKET_BUFFER_SIZE_OS_MAX**

  OS maximum to be used to specify the socket buffer size.
- #define **NDDS_TRANSPORT_UDPV4_SOCKET_BUFFER_SIZE_OS_DEFAULT** **NDDS_TRANSPORT_↩ UDP_SOCKET_BUFFER_SIZE_OS_DEFAULT**

  OS default to be used to specify the socket buffer size.
- #define **NDDS_TRANSPORT_UDPV4_SEND_SOCKET_BUFFER_SIZE_DEFAULT** **NDDS_TRANSPORT_↩ UDP_SEND_SOCKET_BUFFER_SIZE_DEFAULT**

  *Default value for **NDDS_Transport_UDPv4_Property_t::send_socket_buffer_size*** (p. *2058*) *and **NDDS_Transport_↩ UDPv4_WAN_Property_t::send_socket_buffer_size*** (p. *2068*).
- #define **NDDS_TRANSPORT_UDPV4_RECV_SOCKET_BUFFER_SIZE_DEFAULT** **NDDS_TRANSPORT_↩ UDP_RECV_SOCKET_BUFFER_SIZE_DEFAULT**

  *Default value for **NDDS_Transport_UDPv4_Property_t::recv_socket_buffer_size*** (p. *2058*) *and **NDDS_Transport_↩ UDPv4_WAN_Property_t::recv_socket_buffer_size*** (p. *2068*).
- #define **NDDS_TRANSPORT_UDPV4_PAYLOAD_SIZE_MAX** (65507)

  *Maximum value of **NDDS_Transport_Property_t::message_size_max*** (p. *2047*).
- #define **NDDS_TRANSPORT_UDPV4_MESSAGE_SIZE_MAX_DEFAULT** **NDDS_TRANSPORT_UDPV4_↩ PAYLOAD_SIZE_MAX**

  *Maximum value of **NDDS_Transport_Property_t::message_size_max*** (p. *2047*).
- #define **NDDS_TRANSPORT_UDPV4_MULTICAST_TTL_DEFAULT** **NDDS_TRANSPORT_UDP_↩ MULTICAST_TTL_DEFAULT**

  *Default value of **NDDS_Transport_UDPv4_Property_t::multicast_ttl*** (p. *2059*).
- #define **NDDS_TRANSPORT_UDPV4_BLOCKING_NEVER**

  *Value for **NDDS_Transport_UDPv4_Property_t::send_blocking*** (p. *2062*) *to specify non-blocking sockets.*
- #define **NDDS_TRANSPORT_UDPV4_BLOCKING_ALWAYS**

  [default]** Value for **NDDS_Transport_UDPv4_Property_t::send_blocking (p. *2062*) *to specify blocking sockets.*
- #define **NDDS_TRANSPORT_UDPV4_BLOCKING_DEFAULT** NDDS_TRANSPORT_UDP_BLOCKING_↩ DEFAULT

  *Default value for **NDDS_Transport_UDPv4_Property_t::send_blocking*** (p. *2062*) *to specify blocking sockets.*
- #define **NDDS_TRANSPORT_UDPV4_PROPERTY_DEFAULT**

  *Use this to initialize a **NDDS_Transport_UDPv4_Property_t*** (p. *2056*) *structure.*
- #define **NDDS_Transport_UDPv4_string_to_address_cEA** NDDS_Transport_UDP_string_to_address_cEA

  Realization of NDDS_Transport_String_To_Address_Fcn_cEA for IP transports.

Functions

- NDDS_Transport_Plugin * **NDDS_Transport_UDPv4_new** (const struct **NDDS_Transport_UDPv4_Property_t** *property_in)

  Create an instance of a UDPv4 Transport Plugin.
- NDDS_Transport_Plugin * **NDDS_Transport_UDPv4_create** (**NDDS_Transport_Address_t** *default_network_address_out, const struct **DDS_PropertyQosPolicy** *property_in)

  Create an instance of a UDPv4 Transport Plugin, using the PropertyQosPolicy.
- NDDS_Transport_Plugin * **NDDS_Transport_UDPv4_create_from_properties_with_prefix** (**NDDS_Transport_Address_t** *default_network_address_out, const struct **DDS_PropertyQosPolicy** *property_in, const char *propertyPrefix)

  Create an instance of a UDPv4 Transport Plugin, using the PropertyQosPolicy. Same as NDDS_Transport_UDPv4_create but with a prefix for the input properties. It is used to create pluggable transports that use an underlying UDP transport.

7.115.1 Detailed Description

Transport plug-in using UDP/IPv4 (**NDDS_TRANSPORT_CLASSID_UDPv4** (p. 444)) .

This transport plugin uses UDPv4 sockets to send and receive messages. It supports both unicast and multicast communications in a single instance of the plugin. By default, this plugin will use all interfaces that it finds enabled and "UP" at instantiation time to send and receive messages. This transport is installed as a built-in transport plugin with the alias **DDS_TRANSPORTBUILTIN_UDPv4_ALIAS** (p. 644).

You can configure an instance of this plugin to only use unicast or only use multicast, see **NDDS_Transport_UDPv4_Property_t::unicast_enabled** (p. 2059) and **NDDS_Transport_UDPv4_Property_t::multicast_enabled** (p. 2059).

In addition, you can configure an instance of this plugin to selectively use the network interfaces of a node (and restrict a plugin from sending multicast messages on specific interfaces) by specifying the **NDDS_Transport_Property_t::max_interface_count** (p. 2051) and the "white" and "black" lists in the base property's fields (**NDDS_Transport_Property_t::allow_interfaces_list** (p. 2047), **NDDS_Transport_Property_t::deny_interfaces_list** (p. 2048), **NDDS_Transport_Property_t::allow_multicast_interfaces_list** (p. 2049), **NDDS_Transport_Property_t::deny_multicast_interfaces_list** (p. 2050)).

RTI Connext can implicitly create this plugin and register with the **DDSDomainParticipant** (p. 1619) if this transport is specified in **DDS_TransportBuiltinQosPolicy** (p. 1408).

To specify the properties of the builtin UDPv4 transport that is implicitly registered, you can either:

- call **NDDSTransportSupport::set_builtin_transport_property** (p. 2109) or
- specify the predefined property names in **DDS_PropertyQosPolicy** (p. 1260) associated with the **DDSDomainParticipant** (p. 1619). (see **UDPv4 Transport Property Names in Property QoS Policy of Domain Participant** (p. 465)). Builtin transport plugin properties specified in **DDS_PropertyQosPolicy** (p. 1260) always overwrite the ones specified through **NDDSTransportSupport::set_builtin_transport_property()** (p. 2109). The default value is assumed on any unspecified property.

Note that all properties should be set before the transport is implicitly created and registered by RTI Connext. Any properties set after the builtin transport is registered will be ignored. See **Built-in Transport Plugins** (p. 182) for details on when a builtin transport is registered. To explicitly create an instance of this plugin, **NDDS_Transport_UDPv4_new()** (p. 470) should be called. The instance should be registered with RTI Connext, see **NDDSTransportSupport::register_transport** (p. 2104). In some configurations one may have to disable the builtin UDPv4 transport plugin instance (**DDS_TransportBuiltinQosPolicy** (p. 1408), **DDS_TRANSPORTBUILTIN_UDPv4** (p. 644)), to avoid port conflicts with the newly created plugin instance.

7.115.2 UDPv4 Transport Property Names in Property QoS Policy of Domain Participant

The following table lists the predefined property names that can be set in the **DDS_DomainParticipantQos::property** (p. 975) to configure the builtin UDPv4 transport plugin.

Table 7.155 ***Property Names for UDPv4 Transport Plugin***

Property Name	Description
dds.transport.UDPv4.builtin.parent.classid	See **NDDS_Transport_Property_t::classid** (p. 2045) Should be set to "**NDDS_TRANSPORT_CLASSID_UDPv6** (p. 444)"
dds.transport.UDPv4.builtin.parent.address_bit_count	See **NDDS_Transport_Property_t::address_bit_count** (p. 2046)
dds.transport.UDPv4.builtin.parent.properties_bitmap	See **NDDS_Transport_Property_t::properties_bitmap** (p. 2046)
dds.transport.UDPv4.builtin.parent.gather_send_buffer_count_max	See **NDDS_Transport_Property_t::gather_send_buffer_count_max** (p. 2047)
dds.transport.UDPv4.builtin.parent.message_size_max	See **NDDS_Transport_Property_t::message_size_max** (p. 2047)
dds.transport.UDPv4.builtin.parent.allow_interfaces	See **NDDS_Transport_Property_t::allow_interfaces_list** (p. 2047) and **NDDS_Transport_Property_t::allow_interfaces_list_length** (p. 2048). Interfaces should be specified as comma-separated strings, with each comma delimiting an interface. For example, 127.0.0.1,eth0
dds.transport.UDPv4.builtin.parent.deny_interfaces	See **NDDS_Transport_Property_t::deny_interfaces_list** (p. 2048) and **NDDS_Transport_Property_t::deny_interfaces_list_length** (p. 2049). Interfaces should be specified as comma-separated strings, with each comma delimiting an interface. For example: 127.0.0.1,eth0
dds.transport.UDPv4.builtin.parent.allow_multicast_interfaces	See **NDDS_Transport_Property_t::allow_multicast_interfaces_list** (p. 2049) and **NDDS_Transport_Property_t::allow_multicast_interfaces_list_length** (p. 2050). Interfaces should be specified as comma-separated strings, with each comma delimiting an interface. For example: 127.0.0.1,eth0
dds.transport.UDPv4.builtin.parent.deny_multicast_interfaces	See **NDDS_Transport_Property_t::deny_multicast_interfaces_list** (p. 2050) and **NDDS_Transport_Property_t::deny_multicast_interfaces_list_length** (p. 2050). Interfaces should be specified as comma-separated strings, with each comma delimiting an interface. For example: 127.0.0.1,eth0
dds.transport.UDPv4.builtin.parent.max_interface_count	See **NDDS_Transport_Property_t::max_interface_count** (p. 2051)
dds.transport.UDPv4.builtin.parent.thread_name_prefix	See **NDDS_Transport_Property_t::thread_name_prefix** (p. 2051)
dds.transport.UDPv4.builtin.send_socket_buffer_size	See **NDDS_Transport_UDPv4_Property_t::send_socket_buffer_size** (p. 2058)
dds.transport.UDPv4.builtin.recv_socket_buffer_size	See **NDDS_Transport_UDPv4_Property_t::recv_socket_buffer_size** (p. 2058)

Property Name	Description
dds.transport.UDPv4.builtin.unicast_enabled	See **NDDS_Transport_UDPv4_Property_t::unicast_enabled** (p. 2059)
dds.transport.UDPv4.builtin.multicast_enabled	See **NDDS_Transport_UDPv4_Property_t::multicast_enabled** (p. 2059)
dds.transport.UDPv4.builtin.multicast_ttl	See **NDDS_Transport_UDPv4_Property_t::multicast_ttl** (p. 2059)
dds.transport.UDPv4.builtin.multicast_loopback_disabled	See **NDDS_Transport_UDPv4_Property_t::multicast_loopback_disabled** (p. 2060)
dds.transport.UDPv4.builtin.ignore_loopback_interface	See **NDDS_Transport_UDPv4_Property_t::ignore_loopback_interface** (p. 2060)
dds.transport.UDPv4.builtin.ignore_nonrunning_interfaces	See **NDDS_Transport_UDPv4_Property_t::ignore_nonrunning_interfaces** (p. 2061)
dds.transport.UDPv4.builtin.ignore_nonup_interfaces	**[DEPRECATED]** See **NDDS_Transport_UDPv4_Property_t::ignore_nonup_interfaces** (p. 2060)
dds.transport.UDPv4.builtin.no_zero_copy	**[DEPRECATED]** See **NDDS_Transport_UDPv4_Property_t::no_zero_copy** (p. 2061)
dds.transport.UDPv4.builtin.send_blocking	See **NDDS_Transport_UDPv4_Property_t::send_blocking** (p. 2062)
dds.transport.UDPv4.builtin.transport_priority_mask	See **NDDS_Transport_UDPv4_Property_t::transport_priority_mask** (p. 2062)
dds.transport.UDPv4.builtin.transport_priority_mapping_low	See **NDDS_Transport_UDPv4_Property_t::transport_priority_mapping_low** (p. 2063)
dds.transport.UDPv4.builtin.transport_priority_mapping_high	See **NDDS_Transport_UDPv4_Property_t::transport_priority_mapping_high** (p. 2063)
dds.transport.UDPv4.builtin.send_ping	See **NDDS_Transport_UDPv4_Property_t::send_ping** (p. 2063)
dds.transport.UDPv4.builtin.force_interface_poll_detection	See **NDDS_Transport_UDPv4_Property_t::force_interface_poll_detection** (p. 2064)
dds.transport.UDPv4.builtin.interface_poll_period	See **NDDS_Transport_UDPv4_Property_t::interface_poll_period** (p. 2064)
dds.transport.UDPv4.builtin.reuse_multicast_receive_resource	See **NDDS_Transport_UDPv4_Property_t::reuse_multicast_receive_resource** (p. 2064)
dds.transport.UDPv4.builtin.protocol_overhead_max	See **NDDS_Transport_UDPv4_Property_t::protocol_overhead_max** (p. 2064)
dds.transport.UDPv4.builtin.disable_interface_tracking	See **NDDS_Transport_UDPv4_Property_t::disable_interface_tracking** (p. 2065)
dds.transport.UDPv4.builtin.public_address	See **NDDS_Transport_UDPv4_Property_t::public_address** (p. 2065)
dds.transport.UDPv4.builtin.join_multicast_group_timeout	See **NDDS_Transport_UDPv4_Property_t::join_multicast_group_timeout** (p. 2065)
dds.transport.UDPv4.builtin.port_offset	See **NDDS_Transport_UDPv4_Property_t::port_offset** (p. 2066)

See also

NDDSTransportSupport::set_builtin_transport_property() (p. 2109)

7.115.3 Macro Definition Documentation

7.115.3.1 NDDS_TRANSPORT_UDPV4_ADDRESS_BIT_COUNT

```
#define NDDS_TRANSPORT_UDPV4_ADDRESS_BIT_COUNT (32)
```

Default value of **NDDS_Transport_Property_t::address_bit_count** (p. 2046).

7.115.3.2 NDDS_TRANSPORT_UDPV4_WAN_ADDRESS_BIT_COUNT

```
#define NDDS_TRANSPORT_UDPV4_WAN_ADDRESS_BIT_COUNT (128)
```

Default value of **NDDS_Transport_Property_t::address_bit_count** (p. 2046) for UDPv4 asymmetric transport.

7.115.3.3 NDDS_TRANSPORT_UDPV4_PROPERTIES_BITMAP_DEFAULT

```
#define NDDS_TRANSPORT_UDPV4_PROPERTIES_BITMAP_DEFAULT   NDDS_TRANSPORT_UDP_PROPERTIES_BITMAP_↩
DEFAULT
```

Default value of **NDDS_Transport_Property_t::properties_bitmap** (p. 2046).

7.115.3.4 NDDS_TRANSPORT_UDPV4_GATHER_SEND_BUFFER_COUNT_MAX_DEFAULT

```
#define NDDS_TRANSPORT_UDPV4_GATHER_SEND_BUFFER_COUNT_MAX_DEFAULT   NDDS_TRANSPORT_UDP_GATHER_↩
SEND_BUFFER_COUNT_MAX_DEFAULT
```

Default value of **NDDS_Transport_Property_t::gather_send_buffer_count_max** (p. 2047).

This is also the maximum value that can be used when instantiating the UDP transport.

16 is sufficient for RTI Connext, but more may improve discovery and reliable performance. Porting note: find out what the maximum gather buffer count is on your OS!

7.115.3.5 NDDS_TRANSPORT_UDPV4_SOCKET_BUFFER_SIZE_OS_MAX

```
#define NDDS_TRANSPORT_UDPV4_SOCKET_BUFFER_SIZE_OS_MAX   NDDS_TRANSPORT_UDP_SOCKET_BUFFER_SIZE_↩
OS_MAX
```

OS maximum to be used to specify the socket buffer size.

This property, if set, uses the maximum allowable size set in your OS for send and receive socket buffer sizes. If you update the maximum allowable size in your OS setting, then RTI Connext will automatically pick up that change. This property is not supported on all platforms because not all platforms have maximum values for the size of the send and receive buffers for sockets. See the `Platform Notes` to determine if this property is supported on your platform.

7.115.3.6 NDDS_TRANSPORT_UDPV4_SOCKET_BUFFER_SIZE_OS_DEFAULT

```
#define NDDS_TRANSPORT_UDPV4_SOCKET_BUFFER_SIZE_OS_DEFAULT   NDDS_TRANSPORT_UDP_SOCKET_BUFFER_↩
SIZE_OS_DEFAULT
```

OS default to be used to specify the socket buffer size.

7.115.3.7 NDDS_TRANSPORT_UDPV4_SEND_SOCKET_BUFFER_SIZE_DEFAULT

```
#define NDDS_TRANSPORT_UDPV4_SEND_SOCKET_BUFFER_SIZE_DEFAULT   NDDS_TRANSPORT_UDP_SEND_SOCKET_↩
BUFFER_SIZE_DEFAULT
```

Default value for **NDDS_Transport_UDPv4_Property_t::send_socket_buffer_size** (p. 2058) and **NDDS_Transport↩ _UDPv4_WAN_Property_t::send_socket_buffer_size** (p. 2068).

The default value is 2097152 bytes (1048576 bytes on AIX platforms).

7.115.3.8 NDDS_TRANSPORT_UDPV4_RECV_SOCKET_BUFFER_SIZE_DEFAULT

```
#define NDDS_TRANSPORT_UDPV4_RECV_SOCKET_BUFFER_SIZE_DEFAULT   NDDS_TRANSPORT_UDP_RECV_SOCKET_↩
BUFFER_SIZE_DEFAULT
```

Default value for **NDDS_Transport_UDPv4_Property_t::recv_socket_buffer_size** (p. 2058) and **NDDS_Transport↩ _UDPv4_WAN_Property_t::recv_socket_buffer_size** (p. 2068).

The default value is 2097152 bytes (1048576 bytes on AIX platforms).

7.115.3.9 NDDS_TRANSPORT_UDPV4_PAYLOAD_SIZE_MAX

```
#define NDDS_TRANSPORT_UDPV4_PAYLOAD_SIZE_MAX (65507)
```

Maximum value of **NDDS_Transport_Property_t::message_size_max** (p. 2047).

7.115.3.10 NDDS_TRANSPORT_UDPV4_MESSAGE_SIZE_MAX_DEFAULT

```
#define NDDS_TRANSPORT_UDPV4_MESSAGE_SIZE_MAX_DEFAULT   NDDS_TRANSPORT_UDPV4_PAYLOAD_SIZE_MAX
```

Maximum value of **NDDS_Transport_Property_t::message_size_max** (p. 2047).

7.115.3.11 NDDS_TRANSPORT_UDPV4_MULTICAST_TTL_DEFAULT

```
#define NDDS_TRANSPORT_UDPV4_MULTICAST_TTL_DEFAULT   NDDS_TRANSPORT_UDP_MULTICAST_TTL_DEFAULT
```

Default value of **NDDS_Transport_UDPv4_Property_t::multicast_ttl** (p. 2059).

7.115.3.12 NDDS_TRANSPORT_UDPV4_BLOCKING_NEVER

```
#define NDDS_TRANSPORT_UDPV4_BLOCKING_NEVER
```

Value for **NDDS_Transport_UDPv4_Property_t::send_blocking** (p. 2062) to specify non-blocking sockets.

7.115.3.13 NDDS_TRANSPORT_UDPV4_BLOCKING_ALWAYS

```
#define NDDS_TRANSPORT_UDPV4_BLOCKING_ALWAYS
```

[default] Value for **NDDS_Transport_UDPv4_Property_t::send_blocking** (p. 2062) to specify blocking sockets.

7.115.3.14 NDDS_TRANSPORT_UDPV4_BLOCKING_DEFAULT

```
#define NDDS_TRANSPORT_UDPV4_BLOCKING_DEFAULT  NDDS_TRANSPORT_UDP_BLOCKING_DEFAULT
```

Default value for **NDDS_Transport_UDPv4_Property_t::send_blocking** (p. 2062) to specify blocking sockets.

7.115.3.15 NDDS_TRANSPORT_UDPV4_PROPERTY_DEFAULT

```
#define NDDS_TRANSPORT_UDPV4_PROPERTY_DEFAULT
```

Use this to initialize a **NDDS_Transport_UDPv4_Property_t** (p. 2056) structure.

7.115.3.16 NDDS_Transport_UDPv4_string_to_address_cEA

```
#define NDDS_Transport_UDPv4_string_to_address_cEA  NDDS_Transport_UDP_string_to_address_cEA
```

Realization of NDDS_Transport_String_To_Address_Fcn_cEA for IP transports.

Converts a host name string to a IPv4 address.

Parameters

self	NOT USED. May be NULL.
address_out	<<***out***>> *(*p. *433)* The corresponding numerical value in IPv4 format.
address_in	<<***in***>> *(*p. *433)* The name of the IPv4 address. It can be a dot notation name or a host name. If NULL, then the IP address of the localhost will be returned.

See also

NDDS_Transport_String_To_Address_Fcn_cEA for complete documentation.

7.115.4 Function Documentation

7.115.4.1 NDDS_Transport_UDPv4_new()

```
NDDS_Transport_Plugin * NDDS_Transport_UDPv4_new (
            const struct  NDDS_Transport_UDPv4_Property_t * property_in )
```

Create an instance of a UDPv4 Transport Plugin.

An application may create and register multiple instances of this Transport Plugin with RTI Connext. Multiple instances of the plugin may be useful for partitioning the network interfaces across multiple RTI Connext domains. However, note that the underlying transport, the operating system's IP layer, is still a "singleton." For example, if a unicast transport has already bound to a port, and another unicast transport tries to bind to the same port, the second attempt will fail.

The transport plugin honors the interface/multicast "white" and "black" lists specified in the NDDS_Transport_UDP_↩ Property_t::parent:

- **NDDS_Transport_Property_t::allow_interfaces_list** (p. 2047),
- **NDDS_Transport_Property_t::deny_interfaces_list** (p. 2048),
- **NDDS_Transport_Property_t::allow_multicast_interfaces_list** (p. 2049),
- **NDDS_Transport_Property_t::deny_multicast_interfaces_list** (p. 2050)

The format of a string in these lists is assumed to be in standard IPv4 dot notation, possibly containing wildcards. For example:

- 10.10.30.232
- 10.10.∗.∗
- 192.168.1.∗
- etc. Strings not in the correct format will be ignored.

Parameters

property_in	<<***in***>> *(*p. *433)* Desired behavior of this transport. May be NULL for default property.

Returns

A UDPv4 Transport Plugin instance on success; or NULL on failure.

7.115.4.2 NDDS_Transport_UDPv4_create()

```
NDDS_Transport_Plugin * NDDS_Transport_UDPv4_create (
             NDDS_Transport_Address_t * default_network_address_out,
            const struct  DDS_PropertyQosPolicy * property_in )
```

Create an instance of a UDPv4 Transport Plugin, using the PropertyQosPolicy.

An application may create and register multiple instances of this Transport Plugin with RTI Connext. Multiple instances of the plugin may be useful for partitioning the network interfaces across multiple RTI Connext domains. However, note that the underlying transport, the operating system's IP layer, is still a "singleton." For example, if a unicast transport has already bound to a port, and another unicast transport tries to bind to the same port, the second attempt will fail.

The transport plugin honors the interface/multicast "white" and "black" lists specified in the **NDDS_Transport_UDPv4_Property_t::parent** (p. 2058):

- **NDDS_Transport_Property_t::allow_interfaces_list** (p. 2047),
- **NDDS_Transport_Property_t::deny_interfaces_list** (p. 2048),
- **NDDS_Transport_Property_t::allow_multicast_interfaces_list** (p. 2049),
- **NDDS_Transport_Property_t::deny_multicast_interfaces_list** (p. 2050)

The format of a string in these lists is assumed to be in standard IPv4 dot notation, possibly containing wildcards. For example:

- 10.10.30.232
- 10.10.*.*
- 192.168.1.*
- etc. Strings not in the correct format will be ignored.

Parameters

default_network_address_out	<<***out***>> *(*p. *433)* Network address to be used when registering the transport.
property_in	<<***in***>> *(*p. *433)* Desired behavior of this transport as defined in the **DDS_DomainParticipantQos::property** (p. 975).

Returns

A UDPv4 Transport Plugin instance on success; or NULL on failure.

7.115.4.3 NDDS_Transport_UDPv4_create_from_properties_with_prefix()

```
NDDS_Transport_Plugin * NDDS_Transport_UDPv4_create_from_properties_with_prefix (
             NDDS_Transport_Address_t * default_network_address_out,
            const struct  DDS_PropertyQosPolicy * property_in,
            const char * propertyPrefix )
```

Create an instance of a UDPv4 Transport Plugin, using the PropertyQosPolicy. Same as NDDS_Transport_UDPv4_↩ create but with a prefix for the input properties. It is used to create pluggable transports that use an underlying UDP transport.

Parameters

default_network_address_out	<<***out***>> *(*p. *433)* Network address to be used when registering the transport.
property_in	<<***in***>> *(*p. *433)* Desired behavior of this transport as defined in the **DDS_DomainParticipantQos::property** (p. 975).
propertyPrefix	a prefix for the properties. Expected UDP properties have the form prefix.property_name

7.116 Real-Time WAN Transport

Transport plug-in using UDP/IPv4 for WAN communications. (**NDDS_TRANSPORT_CLASSID_UDPv4_WAN** (p. 446)) .

Classes

- struct **NDDS_Transport_UDPv4_WAN_Property_t**

  Configurable IPv4/UDP WAN Transport-Plugin properties.

Macros

- #define **NDDS_TRANSPORT_UDPV4_WAN_MESSAGE_SIZE_MAX_DEFAULT** (1400)

  *Maximum value of **NDDS_Transport_Property_t::message_size_max** (*p. *2047).*
- #define **NDDS_TRANSPORT_UDPV4_WAN_PROPERTY_DEFAULT**

  *Use this to initialize a **NDDS_Transport_UDPv4_WAN_Property_t** (*p. *2067) structure.*

Functions

- NDDS_Transport_Plugin * **NDDS_Transport_UDPv4_WAN_new** (const struct **NDDS_Transport_UDPv4_WAN_Property_t** *property_in)

  Create an instance of a UDPv4_WAN Transport Plugin.
- NDDS_Transport_Plugin * **NDDS_Transport_UDPv4_WAN_create** (**NDDS_Transport_Address_t** *default_network_address_out, const struct **DDS_PropertyQosPolicy** *property_in)

  Create an instance of a UDPv4_WAN Transport Plugin, using the PropertyQosPolicy.
- NDDS_Transport_Plugin * **NDDS_Transport_UDPv4_WAN_create_from_properties_with_prefix** (**NDDS_Transport_Address_t** *default_network_address_out, const struct **DDS_PropertyQosPolicy** *property_in, const char *propertyPrefix)

  Create an instance of a UDPv4_WAN Transport Plugin, using the PropertyQosPolicy. Same as NDDS_Transport_UDPv4_WAN_create but with a prefix for the input properties. It is used to create pluggable transports that use an underlying UDP transport.

7.116.1 Detailed Description

Transport plug-in using UDP/IPv4 for WAN communications. (**NDDS_TRANSPORT_CLASSID_UDPv4_WAN** (p. 446)) .

RTI Real-Time WAN Transport (RWT) is a transport that enables secure, scalable, and high-performance communication over wide area networks (WANs), including public networks.

It extends RTI Connext capabilities to WAN environments. Real-Time WAN Transport uses UDPv4 as the underlying IP transport-layer protocol to better anticipate and adapt to the challenges of diverse network conditions, device mobility, and the dynamic nature of WAN system architectures.

Real-Time WAN Transport, in combination with RTI Cloud Discovery Service (CDS), provides a complete, seamless solution out of the box for WAN connectivity.

This transport is not installed as part of an RTI Connext package; it must be downloaded and installed separately.

Real-Time WAN Transport replaces the transport capabilities of the Secure WAN Transport optionally available with previous RTI Connext releases (prior to 7.0.0), and provides the following capabilities:

- NAT (Network Address Translator) traversal: Ability to communicate between DomainParticipants running in a Local Area Network (LAN) that is behind a NAT-enabled router, and DomainParticipants on the outside of the NAT across a WAN. This functionality is provided in combination with Cloud Discovery Service.

- IP mobility: Support for network transitions and changes in IP addresses in any of the DomainParticipants participating in the communication.

- Security: Secure communications between DomainParticipants using Security Plugins.

Real-Time WAN Transport does not require third-party components, such as STUN servers, or protocols like SIP to handle session establishment. Using a single API and security model, you can leverage the extensive capabilities of the RTI Connext framework and ecosystem, including tools and infrastructure services, even for real-time connectivity from edge to cloud and back in highly distributed systems that communicate across wide area networks.

This transport plugin uses UDPv4 sockets to send and receive messages. It supports unicast communications in a single instance of the plugin. By default, this plugin will use all interfaces that it finds enabled and "UP" at instantiation time to send and receive messages. This transport is installed as a built-in transport plugin with the alias **DDS_↩ TRANSPORTBUILTIN_UDPv4_WAN_ALIAS** (p. 644).

In addition, you can configure an instance of this plugin to selectively use the network interfaces of a node by specifying the **NDDS_Transport_Property_t::max_interface_count** (p. 2051) and the "white" and "black" lists in the base property's fields (**NDDS_Transport_Property_t::allow_interfaces_list** (p. 2047), **NDDS_Transport_Property_t::deny_↩ interfaces_list** (p. 2048)).

RTI Connext can implicitly create this plugin and register with the **DDSDomainParticipant** (p. 1619) if this transport is specified in **DDS_TransportBuiltinQosPolicy** (p. 1408).

To specify the properties of the Real-Time WAN Transport that is implicitly registered, you can either:

- call **NDDSTransportSupport::set_builtin_transport_property** (p. 2109) or
- specify the predefined property names in **DDS_PropertyQosPolicy** (p. 1260) associated with the **DDSDomain↩ Participant** (p. 1619). (see **Real-Time WAN Transport Property** (p. 474)). Builtin transport plugin properties specified in **DDS_PropertyQosPolicy** (p. 1260) always overwrite the ones specified through **NDDSTransport↩ Support::set_builtin_transport_property()** (p. 2109). The default value is assumed on any unspecified property.

Note that all properties should be set before the transport is implicitly created and registered by RTI Connext. Any properties set after the builtin transport is registered will be ignored. See **Built-in Transport Plugins** (p. 182) for details on when a builtin transport is registered. To explicitly create an instance of this plugin, **NDDS_Transport_UDPv4_↩ WAN_new()** (p. 477) should be called. The instance should be registered with RTI Connext, see **NDDSTransport↩ Support::register_transport** (p. 2104). In some configurations one may have to disable the builtin UDPv4 transport plugin instance (**DDS_TransportBuiltinQosPolicy** (p. 1408), **DDS_TRANSPORTBUILTIN_UDPv4_WAN** (p. 644)), to avoid port conflicts with the newly created plugin instance.

For additional details on how to configure and use the Real-Time WAN Transport, see the `Core Libraries User's Manual`.

7.116.2 Real-Time WAN Transport Property

Names in Property QoS Policy of Domain Participant

The following table lists the predefined property names that can be set in the **DDS_DomainParticipantQos::property** (p. 975) to configure the Real-Time WAN Transport plugin.

Table 7.160 ***Property Names for Real-Time WAN Transport Plugin***

Property Name	Description
dds.transport.UDPv4_WAN.builtin.parent.classid	See **NDDS_Transport_Property_t::classid** (p. 2045) Should be set to "**NDDS_TRANSPORT_CLASSID_↩ UDPv4_WAN** (p. 446)"
dds.transport.UDPv4_WAN.builtin.parent.address_bit↩ _count	See **NDDS_Transport_Property_t::address_bit_↩ count** (p. 2046)
dds.transport.UDPv4_WAN.builtin.parent.properties_↩ bitmap	See **NDDS_Transport_Property_t::properties_↩ bitmap** (p. 2046)
dds.transport.UDPv4_WAN.builtin.parent.gather_↩ send_buffer_count_max	See **NDDS_Transport_Property_t::gather_send_↩ buffer_count_max** (p. 2047)
dds.transport.UDPv4_WAN.builtin.parent.message_↩ size_max	See **NDDS_Transport_Property_t::message_size_↩ max** (p. 2047)
dds.transport.UDPv4_WAN.builtin.parent.allow_↩ interfaces	See **NDDS_Transport_Property_t::allow_↩ interfaces_list** (p. 2047) and **NDDS_Transport_↩ Property_t::allow_interfaces_list_length** (p. 2048). Interfaces should be specified as comma-separated strings, with each comma delimiting an interface. For example, 127.0.0.1,eth0
dds.transport.UDPv4_WAN.builtin.parent.deny_↩ interfaces	See **NDDS_Transport_Property_t::deny_interfaces↩ _list** (p. 2048) and **NDDS_Transport_Property_t↩ ::deny_interfaces_list_length** (p. 2049). Interfaces should be specified as comma-separated strings, with each comma delimiting an interface. For example: 127.0.0.1,eth0
dds.transport.UDPv4_WAN.builtin.parent.max_↩ interface_count	See **NDDS_Transport_Property_t::max_interface_↩ count** (p. 2051)
dds.transport.UDPv4_WAN.builtin.parent.thread_↩ name_prefix	See **NDDS_Transport_Property_t::thread_name_↩ prefix** (p. 2051)
dds.transport.UDPv4_WAN.builtin.send_socket_↩ buffer_size	See **NDDS_Transport_UDPv4_WAN_Property_t↩ ::send_socket_buffer_size** (p. 2068)
dds.transport.UDPv4_WAN.builtin.recv_socket_buffer↩ _size	See **NDDS_Transport_UDPv4_WAN_Property_t↩ ::recv_socket_buffer_size** (p. 2068)
dds.transport.UDPv4_WAN.builtin.ignore_loopback_↩ interface	See **NDDS_Transport_UDPv4_WAN_Property_t↩ ::ignore_loopback_interface** (p. 2069)
dds.transport.UDPv4_WAN.builtin.ignore_nonrunning↩ _interfaces	See **NDDS_Transport_UDPv4_WAN_Property_t↩ ::ignore_nonrunning_interfaces** (p. 2070)
dds.transport.UDPv4_WAN.builtin.ignore_nonup_↩ interfaces	**[DEPRECATED]** See **NDDS_Transport_UDPv4_↩ WAN_Property_t::ignore_nonup_interfaces** (p. 2069)
dds.transport.UDPv4_WAN.builtin.no_zero_copy	**[DEPRECATED]** See **NDDS_Transport_UDPv4_↩ WAN_Property_t::no_zero_copy** (p. 2070)
dds.transport.UDPv4_WAN.builtin.send_blocking	See **NDDS_Transport_UDPv4_WAN_Property_t↩ ::send_blocking** (p. 2071)
dds.transport.UDPv4_WAN.builtin.transport_priority_↩ mask	See **NDDS_Transport_UDPv4_WAN_Property_t↩ ::transport_priority_mask** (p. 2071)
dds.transport.UDPv4_WAN.builtin.transport_priority_↩ mapping_low	See **NDDS_Transport_UDPv4_WAN_Property_t↩ ::transport_priority_mapping_low** (p. 2072)
dds.transport.UDPv4_WAN.builtin.transport_priority_↩ mapping_high	See **NDDS_Transport_UDPv4_WAN_Property_t↩ ::transport_priority_mapping_high** (p. 2072)

Property Name	Description
dds.transport.UDPv4_WAN.builtin.send_ping	See **NDDS_Transport_UDPv4_WAN_Property_t::send_ping** (p. 2072)
dds.transport.UDPv4_WAN.builtin.force_interface_poll_detection	See **NDDS_Transport_UDPv4_WAN_Property_t::force_interface_poll_detection** (p. 2073)
dds.transport.UDPv4_WAN.builtin.interface_poll_period	See **NDDS_Transport_UDPv4_WAN_Property_t::interface_poll_period** (p. 2073)
dds.transport.UDPv4_WAN.builtin.protocol_overhead_max	See **NDDS_Transport_UDPv4_WAN_Property_t::protocol_overhead_max** (p. 2073)
dds.transport.UDPv4_WAN.builtin.disable_interface_tracking	See **NDDS_Transport_UDPv4_WAN_Property_t::disable_interface_tracking** (p. 2073)
dds.transport.UDPv4_WAN.builtin.public_address	See **NDDS_Transport_UDPv4_WAN_Property_t::public_address** (p. 2074)
dds.transport.UDPv4_WAN.builtin.comm_ports	See **NDDS_Transport_UDPv4_WAN_Property_t::comm_ports_list** (p. 2074)
dds.transport.UDPv4_WAN.builtin.port_offset	See **NDDS_Transport_UDPv4_WAN_Property_t::port_offset** (p. 2076)
dds.transport.UDPv4_WAN.builtin.binding_ping_period	See **NDDS_Transport_UDPv4_WAN_Property_t::binding_ping_period** (p. 2076)

See also

NDDSTransportSupport::set_builtin_transport_property() (p. 2109)

7.116.3 Macro Definition Documentation

7.116.3.1 NDDS_TRANSPORT_UDPV4_WAN_MESSAGE_SIZE_MAX_DEFAULT

```
#define NDDS_TRANSPORT_UDPV4_WAN_MESSAGE_SIZE_MAX_DEFAULT (1400)
```

Maximum value of **NDDS_Transport_Property_t::message_size_max** (p. 2047).

7.116.3.2 NDDS_TRANSPORT_UDPV4_WAN_PROPERTY_DEFAULT

```
#define NDDS_TRANSPORT_UDPV4_WAN_PROPERTY_DEFAULT
```

Use this to initialize a **NDDS_Transport_UDPv4_WAN_Property_t** (p. 2067) structure.

7.116.4 Function Documentation

7.116.4.1 NDDS_Transport_UDPv4_WAN_new()

```
NDDS_Transport_Plugin * NDDS_Transport_UDPv4_WAN_new (
            const struct  NDDS_Transport_UDPv4_WAN_Property_t * property_in )
```

Create an instance of a UDPv4_WAN Transport Plugin.

An application may create and register multiple instances of this Transport Plugin with RTI Connext. Multiple instances of the plugin may be useful for partitioning the network interfaces across multiple RTI Connext domains. However, note that the underlying transport, the operating system's IP layer, is still a "singleton." For example, if a unicast transport has already bound to a port, and another unicast transport tries to bind to the same port, the second attempt will fail.

The transport plugin honors the interface "white" and "black" lists specified in the NDDS_Transport_UDP_Property_t↩ ::parent:

- **NDDS_Transport_Property_t::allow_interfaces_list** (p. 2047),
- **NDDS_Transport_Property_t::deny_interfaces_list** (p. 2048),

The format of a string in these lists is assumed to be in standard IPv4 dot notation, possibly containing wildcards. For example:

- 10.10.30.232
- 10.10.∗.∗
- 192.168.1.∗
- etc. Strings not in the correct format will be ignored.

Parameters

property↩ _in	*<<**in**>> (p. 433)* Desired behavior of this transport. May be NULL for default property.

Returns

A UDPv4_WAN Transport Plugin instance on success; or NULL on failure.

7.116.4.2 NDDS_Transport_UDPv4_WAN_create()

```
NDDS_Transport_Plugin * NDDS_Transport_UDPv4_WAN_create (
```

```
 NDDS_Transport_Address_t * default_network_address_out,
const struct  DDS_PropertyQosPolicy * property_in )
```

Create an instance of a UDPv4_WAN Transport Plugin, using the PropertyQosPolicy.

An application may create and register multiple instances of this Transport Plugin with RTI Connext. Multiple instances of the plugin may be useful for partitioning the network interfaces across multiple RTI Connext domains. However, note that the underlying transport, the operating system's IP layer, is still a "singleton." For example, if a unicast transport has already bound to a port, and another unicast transport tries to bind to the same port, the second attempt will fail.

The transport plugin honors the interface "white" and "black" lists specified in the **NDDS_Transport_UDPv4_WAN_↩ Property_t::parent** (p. 2068):

- **NDDS_Transport_Property_t::allow_interfaces_list** (p. 2047),
- **NDDS_Transport_Property_t::deny_interfaces_list** (p. 2048),

The format of a string in these lists is assumed to be in standard IPv4 dot notation, possibly containing wildcards. For example:

- 10.10.30.232
- 10.10.∗.∗
- 192.168.1.∗
- etc. Strings not in the correct format will be ignored.

Parameters

default_network_address_out	*<<**out**>> (p. 433)* Network address to be used when registering the transport.
property_in	*<<**in**>> (p. 433)* Desired behavior of this transport as defined in the **DDS_DomainParticipantQos::property** (p. 975).

Returns

A UDPv4_WAN Transport Plugin instance on success; or NULL on failure.

7.116.4.3 NDDS_Transport_UDPv4_WAN_create_from_properties_with_prefix()

```
NDDS_Transport_Plugin * NDDS_Transport_UDPv4_WAN_create_from_properties_with_prefix (
 NDDS_Transport_Address_t * default_network_address_out,
const struct  DDS_PropertyQosPolicy * property_in,
const char * propertyPrefix )
```

Create an instance of a UDPv4_WAN Transport Plugin, using the PropertyQosPolicy. Same as NDDS_Transport_↩ UDPv4_WAN_create but with a prefix for the input properties. It is used to create pluggable transports that use an underlying UDP transport.

Parameters

default_network_address_out	<<***out***>> *(*p. *433)* Network address to be used when registering the transport.
property_in	<<***in***>> *(*p. *433)* Desired behavior of this transport as defined in the **DDS_DomainParticipantQos::property** (p. 975).
propertyPrefix	Prefix for the properties. Expected UDP properties have the form prefix.property_name

7.117 UDPv6 Transport

Transport plug-in using UDP/IPv6 (**NDDS_TRANSPORT_CLASSID_UDPv6** (p. 445)) .

Classes

- struct **NDDS_Transport_UDPv6_Property_t**

  Configurable IPv6/UDP Transport-Plugin properties.

Macros

- #define **NDDS_TRANSPORT_UDPV6_ADDRESS_BIT_COUNT** (128)

  *Default value of **NDDS_Transport_Property_t::address_bit_count** (*p. *2046).*
- #define **NDDS_TRANSPORT_UDPV6_PROPERTIES_BITMAP_DEFAULT** **NDDS_TRANSPORT_UDP_↩ PROPERTIES_BITMAP_DEFAULT**

  *Default value of **NDDS_Transport_Property_t::properties_bitmap** (*p. *2046).*
- #define **NDDS_TRANSPORT_UDPV6_GATHER_SEND_BUFFER_COUNT_MAX_DEFAULT** **NDDS_↩ TRANSPORT_UDP_GATHER_SEND_BUFFER_COUNT_MAX_DEFAULT**

  *Default value of **NDDS_Transport_Property_t::gather_send_buffer_count_max** (*p. *2047).*
- #define **NDDS_TRANSPORT_UDPV6_SOCKET_BUFFER_SIZE_OS_MAX** **NDDS_TRANSPORT_UDP_↩ SOCKET_BUFFER_SIZE_OS_MAX**

  OS maximum to be used to specify the socket buffer size.
- #define **NDDS_TRANSPORT_UDPV6_SOCKET_BUFFER_SIZE_OS_DEFAULT** **NDDS_TRANSPORT_↩ UDP_SOCKET_BUFFER_SIZE_OS_DEFAULT**

  OS default to be used to specify the socket buffer size.
- #define **NDDS_TRANSPORT_UDPV6_SEND_SOCKET_BUFFER_SIZE_DEFAULT** **NDDS_TRANSPORT_↩ UDP_SEND_SOCKET_BUFFER_SIZE_DEFAULT**

  *Default value for **NDDS_Transport_UDPv6_Property_t::send_socket_buffer_size** (*p. *2078).*
- #define **NDDS_TRANSPORT_UDPV6_RECV_SOCKET_BUFFER_SIZE_DEFAULT** **NDDS_TRANSPORT_↩ UDP_RECV_SOCKET_BUFFER_SIZE_DEFAULT**

  *Default value for **NDDS_Transport_UDPv6_Property_t::recv_socket_buffer_size** (*p. *2079).*
- #define **NDDS_TRANSPORT_UDPV6_PAYLOAD_SIZE_MAX** (65487)

  *Maximum value of **NDDS_Transport_Property_t::message_size_max** (*p. *2047).*
- #define **NDDS_TRANSPORT_UDPV6_MESSAGE_SIZE_MAX_DEFAULT** **NDDS_TRANSPORT_UDPV6_↩ PAYLOAD_SIZE_MAX**

  *Default value of **NDDS_Transport_Property_t::message_size_max** (*p. *2047).*

- #define **NDDS_TRANSPORT_UDPV6_MULTICAST_TTL_DEFAULT** **NDDS_TRANSPORT_UDP_↩ MULTICAST_TTL_DEFAULT**

  *Default value of **NDDS_Transport_UDPv6_Property_t::multicast_ttl** (p. 2080).*
- #define **NDDS_TRANSPORT_UDPV6_BLOCKING_NEVER**

  *Value for **NDDS_Transport_UDPv6_Property_t::send_blocking** (p. 2082) to specify non-blocking sockets.*
- #define **NDDS_TRANSPORT_UDPV6_BLOCKING_ALWAYS**

  ***[default]** Value for **NDDS_Transport_UDPv6_Property_t::send_blocking** (p. 2082) to specify blocking sockets.*
- #define **NDDS_Transport_UDPv6_string_to_address_cEA** NDDS_Transport_UDP_string_to_address_cEA

  Realization of NDDS_Transport_String_To_Address_Fcn_cEA for IP transports.

Functions

- NDDS_Transport_Plugin ∗ **NDDS_Transport_UDPv6_new** (const struct **NDDS_Transport_UDPv6_↩ Property_t** ∗property_in)

  Create an instance of a UDPv6 Transport Plugin.
- NDDS_Transport_Plugin ∗ **NDDS_Transport_UDPv6_create** (**NDDS_Transport_Address_t** ∗default_↩ network_address_out, const struct **DDS_PropertyQosPolicy** ∗property_in)

  Create an instance of a UDPv6 Transport Plugin, using the PropertyQosPolicy.
- NDDS_Transport_Plugin ∗ **NDDS_Transport_UDPv6_create_from_properties_with_prefix** (**NDDS_↩ Transport_Address_t** ∗default_network_address_out, const struct **DDS_PropertyQosPolicy** ∗property_in, const char ∗propertyPrefix)

  Create an instance of a UDPv6 Transport Plugin, using the PropertyQosPolicy. Same as NDDS_Transport_UDPv6_create but with a prefix for the input properties. It is used to create pluggable transports that use an underlying UDP transport.

7.117.1 Detailed Description

Transport plug-in using UDP/IPv6 (**NDDS_TRANSPORT_CLASSID_UDPv6** (p. 445)) .

This transport plugin uses UDPv6 sockets to send and receive messages. It supports both unicast and multicast communications in a single instance of the plugin. By default, this plugin will use all interfaces that it finds enabled and "UP" at instantiation time to send and receive messages. This transport is installed as a built-in transport plugin with the alias **DDS_TRANSPORTBUILTIN_UDPv6_ALIAS** (p. 644).

You can configure an instance of this plugin to only use unicast or only use multicast, see **NDDS_Transport_UDPv6↩ _Property_t::unicast_enabled** (p. 2079) and **NDDS_Transport_UDPv6_Property_t::multicast_enabled** (p. 2080).

In addition, you can configure an instance of this plugin to selectively use the network interfaces of a node (and restrict a plugin from sending multicast messages on specific interfaces) by specifying the **NDDS_Transport_↩ Property_t::max_interface_count** (p. 2051) and the "white" and "black" lists in the base property's fields (**NDDS↩ _Transport_Property_t::allow_interfaces_list** (p. 2047), **NDDS_Transport_Property_t::deny_interfaces_list** (p. 2048), **NDDS_Transport_Property_t::allow_multicast_interfaces_list** (p. 2049), **NDDS_Transport_Property_↩ t::deny_multicast_interfaces_list** (p. 2050)).

RTI Connext can implicitly create this plugin and register it with the **DDSDomainParticipant** (p. 1619) if this transport is specified in the **DDS_TransportBuiltinQosPolicy** (p. 1408).

To specify the properties of the builtin UDPv6 transport that is implicitly registered, you can either:

- call **NDDSTransportSupport::set_builtin_transport_property** (p. 2109) or
- specify the predefined property names in **DDS_PropertyQosPolicy** (p. 1260) associated with the **DDSDomain↩ Participant** (p. 1619). (see **UDPv6 Transport Property Names in Property QoS Policy of Domain Participant** (p. 481)). Builtin transport plugin properties specified in **DDS_PropertyQosPolicy** (p. 1260) always overwrite the ones specified through **NDDSTransportSupport::set_builtin_transport_property()** (p. 2109). The default value is assumed on any unspecified property.

Note that all properties should be set before the transport is implicitly created and registered by RTI Connext. Any properties that are set after the builtin transport is registered will be ignored. See **Built-in Transport Plugins** (p. 182) for details on when a builtin transport is registered.

To explicitly create an instance of this plugin, **NDDS_Transport_UDPv6_new()** (p. 486) should be called. The instance should be registered with RTI Connext, see **NDDSTransportSupport::register_transport** (p. 2104). In some configurations, you may have to disable the builtin UDPv6 transport plugin instance (**DDS_TransportBuiltinQosPolicy** (p. 1408), **DDS_TRANSPORTBUILTIN_UDPv6** (p. 644)), to avoid port conflicts with the newly created plugin instance.

7.117.2 UDPv6 Transport Property Names in Property QoS Policy of Domain Participant

The following table lists the predefined property names that can be set in **DDS_PropertyQosPolicy** (p. 1260) of a **DDSDomainParticipant** (p. 1619) to configure the builtin UDPv6 transport plugin.

Table 7.164 ***Property Names for UDPv6 Transport Plugin***

Property Name	Description
dds.transport.UDPv6.builtin.parent.address_bit_count	See **NDDS_Transport_Property_t::address_bit_↩ count** (p. 2046)
dds.transport.UDPv6.builtin.parent.properties_bitmap	See **NDDS_Transport_Property_t::properties_↩ bitmap** (p. 2046)
dds.transport.UDPv6.builtin.parent.gather_send_↩ buffer_count_max	See **NDDS_Transport_Property_t::gather_send_↩ buffer_count_max** (p. 2047)
dds.transport.UDPv6.builtin.parent.message_size_max	See **NDDS_Transport_Property_t::message_size_↩ max** (p. 2047)
dds.transport.UDPv6.builtin.parent.allow_interfaces	See **NDDS_Transport_Property_t::allow_↩ interfaces_list** (p. 2047) and **NDDS_Transport_↩ Property_t::allow_interfaces_list_length** (p. 2048). Interfaces should be specified as comma-separated strings, with each comma delimiting an interface. For example: 127.0.0.1,eth0
dds.transport.UDPv6.builtin.parent.deny_interfaces	See **NDDS_Transport_Property_t::deny_interfaces↩ _list** (p. 2048) and **NDDS_Transport_Property_t↩ ::deny_interfaces_list_length** (p. 2049). Interfaces should be specified as comma-separated strings, with each comma delimiting an interface. For example: 127.0.0.1,eth0
dds.transport.UDPv6.builtin.parent.allow_multicast_↩ interfaces	See **NDDS_Transport_Property_t::allow_multicast↩ _interfaces_list** (p. 2049) and **NDDS_Transport_↩ Property_t::allow_multicast_interfaces_list_length** (p. 2050). Interfaces should be specified as comma-separated strings, with each comma delimiting an interface. For example: 127.0.0.1,eth0

Property Name	Description
dds.transport.UDPv6.builtin.parent.deny_multicast_↩ interfaces	See **NDDS_Transport_Property_t::deny_multicast↩ _interfaces_list** (p. 2050) and **NDDS_Transport_↩ Property_t::deny_multicast_interfaces_list_length** (p. 2050). Interfaces should be specified as comma-separated strings, with each comma delimiting an interface. For example: 127.0.0.1,eth0
dds.transport.UDPv6.builtin.parent.max_interface_count	See **NDDS_Transport_Property_t::max_interface_↩ count** (p. 2051)
dds.transport.UDPv6.builtin.parent.thread_name_prefix	See **NDDS_Transport_Property_t::thread_name_↩ prefix** (p. 2051)
dds.transport.UDPv6.builtin.send_socket_buffer_size	See **NDDS_Transport_UDPv6_Property_t::send_↩ socket_buffer_size** (p. 2078)
dds.transport.UDPv6.builtin.recv_socket_buffer_size	See **NDDS_Transport_UDPv6_Property_t::recv_↩ socket_buffer_size** (p. 2079)
dds.transport.UDPv6.builtin.unicast_enabled	See **NDDS_Transport_UDPv6_Property_t::unicast↩ _enabled** (p. 2079)
dds.transport.UDPv6.builtin.multicast_enabled	See **NDDS_Transport_UDPv6_Property_t↩ ::multicast_enabled** (p. 2080)
dds.transport.UDPv6.builtin.multicast_ttl	See **NDDS_Transport_UDPv6_Property_t↩ ::multicast_ttl** (p. 2080)
dds.transport.UDPv6.builtin.multicast_loopback_↩ disabled	See **NDDS_Transport_UDPv6_Property_t↩ ::multicast_loopback_disabled** (p. 2080)
dds.transport.UDPv6.builtin.ignore_loopback_interface	See **NDDS_Transport_UDPv6_Property_t::ignore_↩ loopback_interface** (p. 2080)
dds.transport.UDPv6.builtin.ignore_nonrunning_↩ interfaces	See **NDDS_Transport_UDPv6_Property_t::ignore_↩ nonrunning_interfaces** (p. 2081)
dds.transport.UDPv6.builtin.no_zero_copy	**[DEPRECATED]** See **NDDS_Transport_UDPv6_↩ Property_t::no_zero_copy** (p. 2081)
dds.transport.UDPv6.builtin.send_blocking	See **NDDS_Transport_UDPv6_Property_t::send_↩ blocking** (p. 2082)
dds.transport.UDPv6.builtin.enable_v4mapped	See **NDDS_Transport_UDPv6_Property_t::enable_↩ v4mapped** (p. 2082)
dds.transport.UDPv6.builtin.transport_priority_mask	See **NDDS_Transport_UDPv6_Property_t↩ ::transport_priority_mask** (p. 2082)
dds.transport.UDPv6.builtin.transport_priority_↩ mapping_low	See **NDDS_Transport_UDPv6_Property_t↩ ::transport_priority_mapping_low** (p. 2083)
dds.transport.UDPv6.builtin.transport_priority_↩ mapping_high	See **NDDS_Transport_UDPv6_Property_t↩ ::transport_priority_mapping_high** (p. 2083)
dds.transport.UDPv6.builtin.send_ping	See **NDDS_Transport_UDPv6_Property_t::send_↩ ping** (p. 2083) interface_poll_detection_kind
dds.transport.UDPv6.builtin.force_interface_poll_↩ detection	See **NDDS_Transport_UDPv6_Property_t::force_↩ interface_poll_detection** (p. 2084)
dds.transport.UDPv6.builtin.interface_poll_period	See **NDDS_Transport_UDPv6_Property_t↩ ::interface_poll_period** (p. 2084)
dds.transport.UDPv6.builtin.reuse_multicast_receive_↩ resource	See **NDDS_Transport_UDPv6_Property_t::reuse_↩ multicast_receive_resource** (p. 2084)
dds.transport.UDPv6.builtin.protocol_overhead_max	See **NDDS_Transport_UDPv6_Property_t↩ ::protocol_overhead_max** (p. 2084)

Property Name	Description
dds.transport.UDPv6.builtin.disable_interface_tracking	See **NDDS_Transport_UDPv6_Property_t::disable_interface_tracking** (p. 2085)
dds.transport.UDPv6.builtin.public_address	See **NDDS_Transport_UDPv6_Property_t::public_address** (p. 2085)
dds.transport.UDPv6.builtin.join_multicast_group_timeout	See **NDDS_Transport_UDPv6_Property_t::join_multicast_group_timeout** (p. 2085)
dds.transport.UDPv6.builtin.port_offset	See **NDDS_Transport_UDPv6_Property_t::port_offset** (p. 2086)

See also

NDDSTransportSupport::set_builtin_transport_property() (p. 2109)

7.117.3 Macro Definition Documentation

7.117.3.1 NDDS_TRANSPORT_UDPV6_ADDRESS_BIT_COUNT

```
#define NDDS_TRANSPORT_UDPV6_ADDRESS_BIT_COUNT (128)
```

Default value of **NDDS_Transport_Property_t::address_bit_count** (p. 2046).

7.117.3.2 NDDS_TRANSPORT_UDPV6_PROPERTIES_BITMAP_DEFAULT

```
#define NDDS_TRANSPORT_UDPV6_PROPERTIES_BITMAP_DEFAULT NDDS_TRANSPORT_UDP_PROPERTIES_BITMAP_DEFAULT
```

Default value of **NDDS_Transport_Property_t::properties_bitmap** (p. 2046).

7.117.3.3 NDDS_TRANSPORT_UDPV6_GATHER_SEND_BUFFER_COUNT_MAX_DEFAULT

```
#define NDDS_TRANSPORT_UDPV6_GATHER_SEND_BUFFER_COUNT_MAX_DEFAULT NDDS_TRANSPORT_UDP_GATHER_SEND_BUFFER_COUNT_MAX_DEFAULT
```

Default value of **NDDS_Transport_Property_t::gather_send_buffer_count_max** (p. 2047).

This is also the maximum value that can be used when instantiating the UDP transport.

16 is sufficient for NDDS, but more may improve discovery and reliable performance. Porting note: find out what the maximum gather buffer count is on your OS!

7.117.3.4 NDDS_TRANSPORT_UDPV6_SOCKET_BUFFER_SIZE_OS_MAX

```
#define NDDS_TRANSPORT_UDPV6_SOCKET_BUFFER_SIZE_OS_MAX   NDDS_TRANSPORT_UDP_SOCKET_BUFFER_SIZE_↩
OS_MAX
```

OS maximum to be used to specify the socket buffer size.

This property, if set, uses the maximum allowable size set in your OS for send and receive socket buffer sizes. If you update the maximum allowable size in your OS setting, then RTI Connext will automatically pick up that change. This property is not supported on all platforms because not all platforms have maximum values for the size of the send and receive buffers for sockets. See the `Platform Notes` to determine if this property is supported on your platform.

7.117.3.5 NDDS_TRANSPORT_UDPV6_SOCKET_BUFFER_SIZE_OS_DEFAULT

```
#define NDDS_TRANSPORT_UDPV6_SOCKET_BUFFER_SIZE_OS_DEFAULT   NDDS_TRANSPORT_UDP_SOCKET_BUFFER_↩
SIZE_OS_DEFAULT
```

OS default to be used to specify the socket buffer size.

7.117.3.6 NDDS_TRANSPORT_UDPV6_SEND_SOCKET_BUFFER_SIZE_DEFAULT

```
#define NDDS_TRANSPORT_UDPV6_SEND_SOCKET_BUFFER_SIZE_DEFAULT   NDDS_TRANSPORT_UDP_SEND_SOCKET_↩
BUFFER_SIZE_DEFAULT
```

Default value for **NDDS_Transport_UDPv6_Property_t::send_socket_buffer_size** (p. 2078).

The default value is 2097152 bytes (1048576 bytes on AIX platforms).

7.117.3.7 NDDS_TRANSPORT_UDPV6_RECV_SOCKET_BUFFER_SIZE_DEFAULT

```
#define NDDS_TRANSPORT_UDPV6_RECV_SOCKET_BUFFER_SIZE_DEFAULT   NDDS_TRANSPORT_UDP_RECV_SOCKET_↩
BUFFER_SIZE_DEFAULT
```

Default value for **NDDS_Transport_UDPv6_Property_t::recv_socket_buffer_size** (p. 2079).

The default value is 2097152 bytes (1048576 bytes on AIX platforms).

7.117.3.8 NDDS_TRANSPORT_UDPV6_PAYLOAD_SIZE_MAX

```
#define NDDS_TRANSPORT_UDPV6_PAYLOAD_SIZE_MAX (65487)
```

Maximum value of **NDDS_Transport_Property_t::message_size_max** (p. 2047).

7.117.3.9 NDDS_TRANSPORT_UDPV6_MESSAGE_SIZE_MAX_DEFAULT

```
#define NDDS_TRANSPORT_UDPV6_MESSAGE_SIZE_MAX_DEFAULT    NDDS_TRANSPORT_UDPV6_PAYLOAD_SIZE_MAX
```

Default value of **NDDS_Transport_Property_t::message_size_max** (p. 2047).

7.117.3.10 NDDS_TRANSPORT_UDPV6_MULTICAST_TTL_DEFAULT

```
#define NDDS_TRANSPORT_UDPV6_MULTICAST_TTL_DEFAULT    NDDS_TRANSPORT_UDP_MULTICAST_TTL_DEFAULT
```

Default value of **NDDS_Transport_UDPv6_Property_t::multicast_ttl** (p. 2080).

7.117.3.11 NDDS_TRANSPORT_UDPV6_BLOCKING_NEVER

```
#define NDDS_TRANSPORT_UDPV6_BLOCKING_NEVER
```

Value for **NDDS_Transport_UDPv6_Property_t::send_blocking** (p. 2082) to specify non-blocking sockets.

7.117.3.12 NDDS_TRANSPORT_UDPV6_BLOCKING_ALWAYS

```
#define NDDS_TRANSPORT_UDPV6_BLOCKING_ALWAYS
```

[default] Value for **NDDS_Transport_UDPv6_Property_t::send_blocking** (p. 2082) to specify blocking sockets.

7.117.3.13 NDDS_Transport_UDPv6_string_to_address_cEA

```
#define NDDS_Transport_UDPv6_string_to_address_cEA  NDDS_Transport_UDP_string_to_address_cEA
```

Realization of NDDS_Transport_String_To_Address_Fcn_cEA for IP transports.

Converts a host name string to a IPv6 address.

Parameters

self	NOT USED. May be NULL.
address_out	<<***out***>> *(p. 433)* The corresponding numerical value in IPv6 format.
address_in	<<***in***>> *(p. 433)* The name of the IPv6 address. It can be a dot notation name or a host name. If NULL, then the IP address of the localhost will be returned.

See also

NDDS_Transport_String_To_Address_Fcn_cEA for complete documentation.

7.117.4 Function Documentation

7.117.4.1 NDDS_Transport_UDPv6_new()

```
NDDS_Transport_Plugin * NDDS_Transport_UDPv6_new (
            const struct  NDDS_Transport_UDPv6_Property_t * property_in )
```

Create an instance of a UDPv6 Transport Plugin.

An application may create and register multiple instances of this Transport Plugin with RTI Connext. Multiple instances of the plugin may be useful for partitioning the network interfaces across multiple RTI Connext domains. However, note that the underlying transport, the operating system's IP layer, is still a "singleton." For example, if a unicast transport has already bound to a port, and another unicast transport tries to bind to the same port, the second attempt will fail.

The transport plugin honors the interface/multicast "white" and "black" lists specified in the **NDDS_Transport_UDPv6_Property_t::parent** (p. 2078):

- **NDDS_Transport_Property_t::allow_interfaces_list** (p. 2047),
- **NDDS_Transport_Property_t::deny_interfaces_list** (p. 2048),
- **NDDS_Transport_Property_t::allow_multicast_interfaces_list** (p. 2049),
- **NDDS_Transport_Property_t::deny_multicast_interfaces_list** (p. 2050)

The format of a string in these lists is assumed to be in standard IPv6 dot notation, possibly containing wildcards. For example:

- 10.10.30.232
- 10.10.*.*
- 192.168.1.*
- etc. Strings not in the correct format will be ignored.

Parameters

property_in	<<***in***>> *(p. 433)* Desired behavior of this transport. May be NULL for default property.

Returns

A UDPv6 Transport Plugin instance on success; or NULL on failure.

7.117.4.2 NDDS_Transport_UDPv6_create()

```
NDDS_Transport_Plugin * NDDS_Transport_UDPv6_create (
             NDDS_Transport_Address_t * default_network_address_out,
            const struct  DDS_PropertyQosPolicy * property_in )
```

Create an instance of a UDPv6 Transport Plugin, using the PropertyQosPolicy.

An application may create and register multiple instances of this Transport Plugin with RTI Connext. Multiple instances of the plugin may be useful for partitioning the network interfaces across multiple RTI Connext domains. However, note that the underlying transport, the operating system's IP layer, is still a "singleton." For example, if a unicast transport has already bound to a port, and another unicast transport tries to bind to the same port, the second attempt will fail.

The transport plugin honors the interface/multicast "white" and "black" lists specified in the **NDDS_Transport_UDPv6_Property_t::parent** (p. 2078):

- **NDDS_Transport_Property_t::allow_interfaces_list** (p. 2047),
- **NDDS_Transport_Property_t::deny_interfaces_list** (p. 2048),
- **NDDS_Transport_Property_t::allow_multicast_interfaces_list** (p. 2049),
- **NDDS_Transport_Property_t::deny_multicast_interfaces_list** (p. 2050)

The format of a string in these lists is assumed to be in standard IPv6 dot notation, possibly containing wildcards. For example:

- *:*:*:*:*:*:*:*
- FE80:aBc::202:*:*
- *:aBC::*2*::2*
- etc. Strings not in the correct format will be ignored.

Parameters

default_network_address_out	<<***out***>> *(p. 433)* Network address to be used when registering the transport.
property_in	<<***in***>> *(p. 433)* Desired behavior of this transport as defined in the **DDS_DomainParticipantQos::property** (p. 975).

Returns

A UDPv6 Transport Plugin instance on success; or NULL on failure.

7.117.4.3 NDDS_Transport_UDPv6_create_from_properties_with_prefix()

```
NDDS_Transport_Plugin * NDDS_Transport_UDPv6_create_from_properties_with_prefix (
             NDDS_Transport_Address_t * default_network_address_out,
            const struct  DDS_PropertyQosPolicy * property_in,
            const char * propertyPrefix )
```

Create an instance of a UDPv6 Transport Plugin, using the PropertyQosPolicy. Same as NDDS_Transport_UDPv6_↩ create but with a prefix for the input properties. It is used to create pluggable transports that use an underlying UDP transport.

Parameters

default_network_address_out	<<***out***>> *(*p. *433)* Network address to be used when registering the transport.
property_in	<<***in***>> *(*p. *433)* Desired behavior of this transport as defined in the **DDS_DomainParticipantQos::property** (p. 975).
propertyPrefix	a prefix for the properties. Expected UDP properties have the form prefix.property_name

7.118 AsyncWaitSet

<<***extension***>> *(*p. *432)* A specialization of **DDSWaitSet** (p. 1897) that provides a mechanism to perform the wait asynchronously and uses a thread pool to dispatch the attached active **DDSCondition** (p. 1544).

Classes

- struct **DDS_AsyncWaitSetProperty_t**

  *Specifies the **DDSAsyncWaitSet** (*p. *1528) behavior.*
- class **DDSAsyncWaitSetListener**

  <<***interface***>> *(*p. *433) Listener for receiving event notifications related to the thread pool of the **DDSAsyncWaitSet** (*p. *1528).*
- class **DDSAsyncWaitSetCompletionToken**

  <<***interface***>> *(*p. *433) Implementation of the completion token role element of the asynchronous completion token pattern that is part of the **DDSAsyncWaitSet** (*p. *1528) behavior.*
- class **DDSAsyncWaitSet**

  *A class for dispatching **DDSCondition** (*p. *1544) objects using separate threads of execution. You can see this class as an extension of a **DDSWaitSet** (*p. *1897) that allows asynchronously waiting for the attached **DDSCondition** (*p. *1544) objects to trigger and provide a notification by calling **DDSCondition::dispatch** (*p. *1546).*
- class **DDSDataReaderStatusConditionHandler**

  <<***interface***>> *(*p. *433) Realization of a **DDSConditionHandler** (*p. *1546) that handles the status of a **DDSDataReader** (*p. *1556).*

Variables

- const struct **DDS_AsyncWaitSetProperty_t DDS_ASYNC_WAITSET_PROPERTY_DEFAULT**

  *Constant that defines the default property for a **DDSAsyncWaitSet** (p. 1528).*
- static **DDSAsyncWaitSetCompletionToken** ∗const **DDSAsyncWaitSet::COMPLETION_TOKEN_USE_IMPLICIT_AND_WAIT**

  *For the operations that allow an **DDSAsyncWaitSetCompletionToken** (p. 1541), this sentinel can be provided to indicate an **DDSAsyncWaitSet** (p. 1528) to use the implicit completion token and wait on it for request completion.*
- static **DDSAsyncWaitSetCompletionToken** ∗const **DDSAsyncWaitSet::COMPLETION_TOKEN_IGNORE**

  *For the operations that allow an **DDSAsyncWaitSetCompletionToken** (p. 1541), this sentinel can be provided to indicate an **DDSAsyncWaitSet** (p. 1528) to perform the action associating a 'null' completion token.*

7.118.1 Detailed Description

<<***extension***>> *(p. 432)* A specialization of **DDSWaitSet** (p. 1897) that provides a mechanism to perform the wait asynchronously and uses a thread pool to dispatch the attached active **DDSCondition** (p. 1544).

This class is a realization of the `Proactor` pattern applied to WaitSets and Conditions that provide a powerful component for your application process events leveraging concurrency.

7.118.2 Variable Documentation

7.118.2.1 DDS_ASYNC_WAITSET_PROPERTY_DEFAULT

`const struct` **DDS_AsyncWaitSetProperty_t** `DDS_ASYNC_WAITSET_PROPERTY_DEFAULT`

Constant that defines the default property for a **DDSAsyncWaitSet** (p. 1528).

Value: `wait_set_property` = **DDS_WaitSetProperty_t_INITIALIZER** (p. 670), `thread_pool_size` = 1, `thread_settings` DDS_THREAD_SETTINGS_DEFAULT, `thread_base_name` = NULL `wait_timeout` = **DDS_DURATION_INFINITE** (p. 522) `level` = 1

7.118.2.2 COMPLETION_TOKEN_USE_IMPLICIT_AND_WAIT

DDSAsyncWaitSetCompletionToken∗ `const DDSAsyncWaitSet::COMPLETION_TOKEN_USE_IMPLICIT_AND_WAIT` `[static]`

For the operations that allow an **DDSAsyncWaitSetCompletionToken** (p. 1541), this sentinel can be provided to indicate an **DDSAsyncWaitSet** (p. 1528) to use the implicit completion token and wait on it for request completion.

If this sentinel is used, the **DDSAsyncWaitSet** (p. 1528) will use the completion token associated with the calling thread and will wait with an infinite timeout until the request completes successfully.

7.118.2.3 COMPLETION_TOKEN_IGNORE

```
DDSAsyncWaitSetCompletionToken* const DDSAsyncWaitSet::COMPLETION_TOKEN_IGNORE  [static]
```

For the operations that allow an **DDSAsyncWaitSetCompletionToken** (p. 1541), this sentinel can be provided to indicate an **DDSAsyncWaitSet** (p. 1528) to perform the action associating a 'null' completion token.

This sentinel is a realization of the null object pattern of an **DDSAsyncWaitSetCompletionToken** (p. 1541). If this object is provided to a **DDSAsyncWaitSet** (p. 1528) operation, the resulting operation request will be associated with a 'null' completion token that behaves as no-op.

When 'nul' completion token is provided, the **DDSAsyncWaitSet** (p. 1528) operation returns immediately after it issues the internal request, and there is no mean for your application to wait for the request to complete.

You can use this sentinel when your application can operate on an **DDSAsyncWaitSet** (p. 1528) without needing to known when operation requests complete or your application uses other strategies to synchronize resources.

This sentinel is also useful when you need to perform operations on **DDSAsyncWaitSet** (p. 1528) from within one of the threads from its thread pool, where waiting on a valid **DDSAsyncWaitSetCompletionToken** (p. 1541) is forbidden.

7.119 Participant Built-in Topics

Builtin topic for accessing information about the DomainParticipants discovered by RTI Connext.

Classes

- struct **DDS_ParticipantBuiltinTopicData**

  Entry created when a DomainParticipant object is discovered.
- struct **DDS_ParticipantBuiltinTopicDataSeq**

  Instantiates ***FooSeq*** *(p. 1965) < **DDS_ParticipantBuiltinTopicData** (p. 1229) > .*
- class **DDSParticipantBuiltinTopicDataTypeSupport**

  Instantiates `TypeSupport` *< **DDS_ParticipantBuiltinTopicData** (p. 1229) > .*
- class **DDSParticipantBuiltinTopicDataDataReader**

  Instantiates `DataReader` *< **DDS_ParticipantBuiltinTopicData** (p. 1229) > .*

Typedefs

- typedef struct **DDS_ParticipantBuiltinTopicData** **DDS_ParticipantBuiltinTopicData**

  Entry created when a DomainParticipant object is discovered.

Variables

- const char * **DDS_PARTICIPANT_TOPIC_NAME**

  Participant topic name.

7.119.1 Detailed Description

Builtin topic for accessing information about the DomainParticipants discovered by RTI Connext.

7.119.2 Typedef Documentation

7.119.2.1 DDS_ParticipantBuiltinTopicData

```
typedef struct DDS_ParticipantBuiltinTopicData DDS_ParticipantBuiltinTopicData
```

Entry created when a DomainParticipant object is discovered.

Data associated with the built-in topic **DDS_PARTICIPANT_TOPIC_NAME** (p. 491). It contains QoS policies and additional information that apply to the remote **DDSDomainParticipant** (p. 1619).

See also

DDS_PARTICIPANT_TOPIC_NAME (p. 491)

DDSParticipantBuiltinTopicDataDataReader (p. 1816)

7.119.3 Variable Documentation

7.119.3.1 DDS_PARTICIPANT_TOPIC_NAME

```
const char* DDS_PARTICIPANT_TOPIC_NAME [extern]
```

Participant topic name.

Topic name of **DDSParticipantBuiltinTopicDataDataReader** (p. 1816)

See also

DDS_ParticipantBuiltinTopicData (p. 1229)

DDSParticipantBuiltinTopicDataDataReader (p. 1816)

7.120 Topic Built-in Topics

Classes

- struct **DDS_TopicBuiltinTopicData**

  Entry created when a Topic object discovered.
- struct **DDS_TopicBuiltinTopicDataSeq**

  Instantiates ***FooSeq*** *(p. 1965) < **DDS_TopicBuiltinTopicData** (p. 1391) > .*
- class **DDSTopicBuiltinTopicDataTypeSupport**

  Instantiates `TypeSupport` *< **DDS_TopicBuiltinTopicData** (p. 1391) > .*
- class **DDSTopicBuiltinTopicDataDataReader**

  Instantiates `DataReader` *< **DDS_TopicBuiltinTopicData** (p. 1391) > .*

Typedefs

- typedef struct **DDS_TopicBuiltinTopicData** **DDS_TopicBuiltinTopicData**

  Entry created when a Topic object discovered.

Variables

- const char * **DDS_TOPIC_TOPIC_NAME**

  Topic topic name.

7.120.1 Detailed Description

Builtin topic for accessing information about the Topics discovered by RTI Connext.

7.120.2 Typedef Documentation

7.120.2.1 DDS_TopicBuiltinTopicData

`typedef struct` **`DDS_TopicBuiltinTopicData`** **`DDS_TopicBuiltinTopicData`**

Entry created when a Topic object discovered.

Data associated with the built-in topic **DDS_TOPIC_TOPIC_NAME** (p. 493). It contains QoS policies and additional information that apply to the remote **DDSTopic** (p. 1885).

Note: The **DDS_TopicBuiltinTopicData** (p. 1391) built-in topic is meant to convey information about discovered Topics. This Topic's samples are not propagated in a separate packet on the wire. Instead, the data is sent as part of the information carried by other built-in topics (**DDS_PublicationBuiltinTopicData** (p. 1264) and **DDS_Subscription↩ BuiltinTopicData** (p. 1370)). Therefore TopicBuiltinTopicData DataReaders will not receive any data.

See also

DDS_TOPIC_TOPIC_NAME (p. 493)

DDSTopicBuiltinTopicDataDataReader (p. 1890)

7.120.3 Variable Documentation

7.120.3.1 DDS_TOPIC_TOPIC_NAME

```
const char* DDS_TOPIC_TOPIC_NAME  [extern]
```

Topic topic name.

Topic name of **DDSTopicBuiltinTopicDataDataReader** (p. 1890)

See also

DDS_TopicBuiltinTopicData (p. 1391)

DDSTopicBuiltinTopicDataDataReader (p. 1890)

7.121 Publication Built-in Topics

Builtin topic for accessing information about the Publications discovered by RTI Connext.

Classes

- struct **DDS_PublicationBuiltinTopicData**

  *Entry created when a **DDSDataWriter** (p. 1589) is discovered in association with its Publisher.*
- struct **DDS_PublicationBuiltinTopicDataSeq**

  *Instantiates **FooSeq** (p. 1965) < **DDS_PublicationBuiltinTopicData** (p. 1264) > .*
- class **DDSPublicationBuiltinTopicDataTypeSupport**

  *Instantiates TypeSupport < **DDS_PublicationBuiltinTopicData** (p. 1264) > .*
- class **DDSPublicationBuiltinTopicDataDataReader**

  *Instantiates DataReader < **DDS_PublicationBuiltinTopicData** (p. 1264) > .*

Typedefs

- typedef struct **DDS_PublicationBuiltinTopicData** **DDS_PublicationBuiltinTopicData**

  *Entry created when a **DDSDataWriter** (p. 1589) is discovered in association with its Publisher.*

Variables

- const char * **DDS_PUBLICATION_TOPIC_NAME**

  Publication topic name.

7.121.1 Detailed Description

Builtin topic for accessing information about the Publications discovered by RTI Connext.

7.121.2 Typedef Documentation

7.121.2.1 DDS_PublicationBuiltinTopicData

```
typedef struct DDS_PublicationBuiltinTopicData DDS_PublicationBuiltinTopicData
```

Entry created when a **DDSDataWriter** (p. 1589) is discovered in association with its Publisher.

Data associated with the built-in topic **DDS_PUBLICATION_TOPIC_NAME** (p. 494). It contains QoS policies and additional information that apply to the remote **DDSDataWriter** (p. 1589) the related **DDSPublisher** (p. 1818).

See also

DDS_PUBLICATION_TOPIC_NAME (p. 494)

DDSPublicationBuiltinTopicDataDataReader (p. 1817)

7.121.3 Variable Documentation

7.121.3.1 DDS_PUBLICATION_TOPIC_NAME

```
const char* DDS_PUBLICATION_TOPIC_NAME [extern]
```

Publication topic name.

Topic name of **DDSPublicationBuiltinTopicDataDataReader** (p. 1817)

See also

DDS_PublicationBuiltinTopicData (p. 1264)

DDSPublicationBuiltinTopicDataDataReader (p. 1817)

7.122 Subscription Built-in Topics

Builtin topic for accessing information about the Subscriptions discovered by RTI Connext.

Classes

- struct **DDS_SubscriptionBuiltinTopicData**

  *Entry created when a **DDSDataReader** (p. 1556) is discovered in association with its Subscriber.*
- struct **DDS_SubscriptionBuiltinTopicDataSeq**

  *Instantiates **FooSeq** (p. 1965) < **DDS_SubscriptionBuiltinTopicData** (p. 1370) > .*
- class **DDSSubscriptionBuiltinTopicDataTypeSupport**

  *Instantiates TypeSupport < **DDS_SubscriptionBuiltinTopicData** (p. 1370) > .*
- class **DDSSubscriptionBuiltinTopicDataDataReader**

  *Instantiates DataReader < **DDS_SubscriptionBuiltinTopicData** (p. 1370) > .*

Typedefs

- typedef struct **DDS_SubscriptionBuiltinTopicData** **DDS_SubscriptionBuiltinTopicData**

  *Entry created when a **DDSDataReader** (p. 1556) is discovered in association with its Subscriber.*

Variables

- const char * **DDS_SUBSCRIPTION_TOPIC_NAME**

  Subscription topic name.

7.122.1 Detailed Description

Builtin topic for accessing information about the Subscriptions discovered by RTI Connext.

7.122.2 Typedef Documentation

7.122.2.1 DDS_SubscriptionBuiltinTopicData

```
typedef struct DDS_SubscriptionBuiltinTopicData DDS_SubscriptionBuiltinTopicData
```

Entry created when a **DDSDataReader** (p. 1556) is discovered in association with its Subscriber.

Data associated with the built-in topic **DDS_SUBSCRIPTION_TOPIC_NAME** (p. 496). It contains QoS policies and additional information that apply to the remote **DDSDataReader** (p. 1556) the related **DDSSubscriber** (p. 1860).

See also

DDS_SUBSCRIPTION_TOPIC_NAME (p. 496)

DDSSubscriptionBuiltinTopicDataDataReader (p. 1882)

7.122.3 Variable Documentation

7.122.3.1 DDS_SUBSCRIPTION_TOPIC_NAME

```
const char* DDS_SUBSCRIPTION_TOPIC_NAME  [extern]
```

Subscription topic name.

Topic name of **DDSSubscriptionBuiltinTopicDataDataReader** (p. 1882)

See also

DDS_SubscriptionBuiltinTopicData (p. 1370)

DDSSubscriptionBuiltinTopicDataDataReader (p. 1882)

7.123 ServiceRequest Built-in Topic

Builtin topic for accessing requests from different services within RTI Connext.

Classes

- struct **DDS_ServiceRequest**

  A request coming from one of the built-in services.
- struct **DDS_ServiceRequestSeq**

  Instantiates ***FooSeq*** *(p. 1965) < **DDS_ServiceRequest** (p. 1359) > .*
- class **DDSServiceRequestTypeSupport**

  Instantiates `TypeSupport` *< **DDS_ServiceRequest** (p. 1359) > .*
- class **DDSServiceRequestDataReader**

  Instantiates `DataReader` *< **DDS_ServiceRequest** (p. 1359) > .*

Typedefs

- typedef struct **DDS_ServiceRequest** **DDS_ServiceRequest**

  A request coming from one of the built-in services.

Variables

- const **DDS_Long DDS_UNKNOWN_SERVICE_REQUEST_ID**

  An invalid Service Id.
- const **DDS_Long DDS_TOPIC_QUERY_SERVICE_REQUEST_ID**

  *Service Id for the **DDSTopicQuery** (p. 1895) Service.*
- const **DDS_Long DDS_LOCATOR_REACHABILITY_SERVICE_REQUEST_ID**

  Service Id for the Locator Reachability Service.
- const **DDS_Long DDS_INSTANCE_STATE_SERVICE_REQUEST_ID**

  Service Id for the Instance State Request service.
- const **DDS_Long DDS_MONITORING_LIBRARY_COMMAND_SERVICE_REQUEST_ID**

  Service Id for RTI Monitoring Library 2.0 Command Service Request.
- const **DDS_Long DDS_MONITORING_LIBRARY_REPLY_SERVICE_REQUEST_ID**

  Service Id for RTI Monitoring Library 2.0 Reply Service Requests.
- const char * **DDS_SERVICE_REQUEST_TOPIC_NAME**

  Service Request topic name.

7.123.1 Detailed Description

Builtin topic for accessing requests from different services within RTI Connext.

Currently, the **DDSTopicQuery** (p. 1895), Locator Reachability Instance State Consistency and Controlability (part of Observability) all rely on this topic.

See also

Topic Queries (p. 155) for an explanation of how TopicQueries use ServiceRequests and how you can access the ServiceRequests for debugging purposes in the section **The Built-in ServiceRequest DataReader** (p. 158).

7.123.2 Typedef Documentation

7.123.2.1 DDS_ServiceRequest

```
typedef struct DDS_ServiceRequest DDS_ServiceRequest
```

A request coming from one of the built-in services.

Data associated with the built-in topic **DDS_SERVICE_REQUEST_TOPIC_NAME** (p. 499). It contains service-specific information.

See also

DDS_SERVICE_REQUEST_TOPIC_NAME (p. 499)

DDSParticipantBuiltinTopicDataDataReader (p. 1816)

7.123.3 Variable Documentation

7.123.3.1 DDS_UNKNOWN_SERVICE_REQUEST_ID

```
const DDS_Long DDS_UNKNOWN_SERVICE_REQUEST_ID [extern]
```

An invalid Service Id.

7.123.3.2 DDS_TOPIC_QUERY_SERVICE_REQUEST_ID

```
const DDS_Long DDS_TOPIC_QUERY_SERVICE_REQUEST_ID [extern]
```

Service Id for the **DDSTopicQuery** (p. 1895) Service.

7.123.3.3 DDS_LOCATOR_REACHABILITY_SERVICE_REQUEST_ID

```
const DDS_Long DDS_LOCATOR_REACHABILITY_SERVICE_REQUEST_ID [extern]
```

Service Id for the Locator Reachability Service.

7.123.3.4 DDS_INSTANCE_STATE_SERVICE_REQUEST_ID

```
const DDS_Long DDS_INSTANCE_STATE_SERVICE_REQUEST_ID [extern]
```

Service Id for the Instance State Request service.

7.123.3.5 DDS_MONITORING_LIBRARY_COMMAND_SERVICE_REQUEST_ID

```
const DDS_Long DDS_MONITORING_LIBRARY_COMMAND_SERVICE_REQUEST_ID [extern]
```

Service Id for RTI Monitoring Library 2.0 Command Service Request.

RTI Monitoring Library 2.0 remote commands are sent using this service.

7.123.3.6 DDS_MONITORING_LIBRARY_REPLY_SERVICE_REQUEST_ID

```
const  DDS_Long DDS_MONITORING_LIBRARY_REPLY_SERVICE_REQUEST_ID  [extern]
```

Service Id for RTI Monitoring Library 2.0 Reply Service Requests.

Replies to RTI Monitoring Library 2.0 remote commands are sent using this service.

7.123.3.7 DDS_SERVICE_REQUEST_TOPIC_NAME

```
const char* DDS_SERVICE_REQUEST_TOPIC_NAME  [extern]
```

Service Request topic name.

Topic name of the **DDSServiceRequestDataReader** (p. 1845)

See also

DDS_ServiceRequest (p. 1359)

DDSServiceRequestDataReader (p. 1845)

7.124 Common types and functions

Types and functions related to the built-in topics.

Classes

- struct **DDS_Locator_t**

  *<<**extension**>> (p. 213) Type used to represent the addressing information needed to send a message to an RTPS Endpoint using one of the supported transports.*
- struct **DDS_LocatorSeq**

  *Declares IDL `sequence` < **DDS_Locator_t** (p. 1184) >*
- struct **DDS_ProtocolVersion_t**

  *<<**extension**>> (p. 432) Type used to represent the version of the RTPS protocol.*
- struct **DDS_VendorId_t**

  *<<**extension**>> (p. 432) Type used to represent the vendor of the service implementing the RTPS protocol.*
- struct **DDS_ProductVersion_t**

  *<<**extension**>> (p. 432) Type used to represent the current version of RTI Connext.*
- struct **DDS_TransportInfo_t**

  Contains the class_id and message_size_max of an installed transport.
- struct **DDS_TransportInfoSeq**

  *Instantiates **FooSeq** (p. 1965) < **DDS_TransportInfo_t** (p. 1409) > .*
- struct **DDS_BuiltinTopicKey_t**

  The key type of the built-in topic types.
- struct **DDS_ContentFilterProperty_t**

  *<<**extension**>> (p. 432) Type used to provide all the required information to enable content filtering.*

Macros

- #define **DDS_LOCATOR_ADDRESS_LENGTH_MAX** 16

  Declares length of address field in locator.
- #define **DDS_PROTOCOLVERSION_1_0** { 1, 0 }

  The protocol version 1.0.
- #define **DDS_PROTOCOLVERSION_1_1** { 1, 1 }

  The protocol version 1.1.
- #define **DDS_PROTOCOLVERSION_1_2** { 1, 2 }

  The protocol version 1.2.
- #define **DDS_PROTOCOLVERSION_2_0** { 2, 0 }

  The protocol version 2.0.
- #define **DDS_PROTOCOLVERSION_2_1** { 2, 1 }

  The protocol version 2.1.
- #define **DDS_PROTOCOLVERSION** { 2, 1 }

  The most recent protocol version. Currently 2.1.
- #define **DDS_VENDOR_ID_LENGTH_MAX** 2

  Length of vendor id.
- #define **DDS_PRODUCTVERSION_UNKNOWN**

  The value used when the product version is unknown.

Typedefs

- typedef struct **DDS_Locator_t** **DDS_Locator_t**

  *<<**extension**>> (p. 213) Type used to represent the addressing information needed to send a message to an RTPS Endpoint using one of the supported transports.*
- typedef struct **DDS_ProtocolVersion_t** **DDS_ProtocolVersion_t**

  *<<**extension**>> (p. 432) Type used to represent the version of the RTPS protocol.*
- typedef struct **DDS_BuiltinTopicKey_t** **DDS_BuiltinTopicKey_t**

  The key type of the built-in topic types.

Functions

- **DDS_Boolean** **DDS_BuiltinTopicKey_equals** (const **DDS_BuiltinTopicKey_t** *self, const **DDS_Builtin↩TopicKey_t** *other)

  Compares this Key with another Key for equality.
- void **DDS_BuiltinTopicKey_copy** (**DDS_BuiltinTopicKey_t** *dst, const **DDS_BuiltinTopicKey_t** *src)

  Copies another Key into this Key.
- void **DDS_BuiltinTopicKey_to_guid** (const **DDS_BuiltinTopicKey_t** *self, **DDS_GUID_t** *dst)

  Converts this Key into a GUID.
- void **DDS_BuiltinTopicKey_from_guid** (**DDS_BuiltinTopicKey_t** *self, const **DDS_GUID_t** *src)

  Initializes this Key from the input GUID.
- **DDS_Boolean** **DDS_BuiltinTopicKey_to_instance_handle** (const **DDS_BuiltinTopicKey_t** *self, **DDS_↩InstanceHandle_t** *dst)

  Converts this Key into an InstanceHandle.
- **DDS_Boolean** **DDS_BuiltinTopicKey_from_instance_handle** (**DDS_BuiltinTopicKey_t** *self, const **DDS↩_InstanceHandle_t** *src)

  Initializes this Key from the input InstanceHandle.

Variables

- const struct **DDS_Locator_t DDS_LOCATOR_INVALID**

  An invalid locator.
- const **DDS_Long DDS_LOCATOR_KIND_INVALID**

  Locator of this kind is invalid.
- const **DDS_UnsignedLong DDS_LOCATOR_PORT_INVALID**

  An invalid port.
- const **DDS_Octet DDS_LOCATOR_ADDRESS_INVALID** [**DDS_LOCATOR_ADDRESS_LENGTH_MAX**]

  An invalid address.
- const **DDS_Long DDS_LOCATOR_KIND_UDPv4**

  A locator for a UDPv4 address.
- const **DDS_Long DDS_LOCATOR_KIND_UDPv4_WAN**

  A locator for a UDPv4 asymmetric transport address.
- const **DDS_Long DDS_LOCATOR_KIND_SHMEM**

  A locator for an address acessed via shared memory.
- const **DDS_Long DDS_LOCATOR_KIND_SHMEM_510**

  A locator for an address acessed via shared memory for RTI Connext 5.1.0 and earlier.
- const **DDS_Long DDS_LOCATOR_KIND_UDPv6**

  A locator for a UDPv6 address.
- const **DDS_Long DDS_LOCATOR_KIND_UDPv6_510**

  A locator for a UDPv6 address for RTI Connext 5.1.0 and earlier.
- const **DDS_Long DDS_LOCATOR_KIND_RESERVED**

  Locator of this kind is reserved.
- DDS_BUILTIN_TOPIC_KEY_TYPE_NATIVE **DDS_BuiltinTopicKey_t::value** [DDS_BUILTIN_TOPIC_KEY_↩ TYPE_NATIVE_LENGTH]

  *An array of four integers that uniquely represents a remote **DDSEntity** (p. 1731).*
- const struct **DDS_BuiltinTopicKey_t DDS_BUILTIN_TOPIC_KEY_UNKNOWN**

  Constant representing an unknown value.

7.124.1 Detailed Description

Types and functions related to the built-in topics.

7.124.2 Macro Definition Documentation

7.124.2.1 DDS_LOCATOR_ADDRESS_LENGTH_MAX

```
#define DDS_LOCATOR_ADDRESS_LENGTH_MAX 16
```

Declares length of address field in locator.

7.124.2.2 DDS_PROTOCOLVERSION_1_0

```
#define DDS_PROTOCOLVERSION_1_0 { 1, 0 }
```

The protocol version 1.0.

7.124.2.3 DDS_PROTOCOLVERSION_1_1

```
#define DDS_PROTOCOLVERSION_1_1 { 1, 1 }
```

The protocol version 1.1.

7.124.2.4 DDS_PROTOCOLVERSION_1_2

```
#define DDS_PROTOCOLVERSION_1_2 { 1, 2 }
```

The protocol version 1.2.

7.124.2.5 DDS_PROTOCOLVERSION_2_0

```
#define DDS_PROTOCOLVERSION_2_0 { 2, 0 }
```

The protocol version 2.0.

7.124.2.6 DDS_PROTOCOLVERSION_2_1

```
#define DDS_PROTOCOLVERSION_2_1 { 2, 1 }
```

The protocol version 2.1.

7.124.2.7 DDS_PROTOCOLVERSION

```
#define DDS_PROTOCOLVERSION { 2, 1 }
```

The most recent protocol version. Currently 2.1.

7.124.2.8 DDS_VENDOR_ID_LENGTH_MAX

```
#define DDS_VENDOR_ID_LENGTH_MAX 2
```

Length of vendor id.

7.124.2.9 DDS_PRODUCTVERSION_UNKNOWN

```
#define DDS_PRODUCTVERSION_UNKNOWN
```

Value:

```
  { \
  DDS_PRODUCTVERSION_MAJOR_UNKNOWN, \
  DDS_PRODUCTVERSION_MINOR_UNKNOWN, \
  DDS_PRODUCTVERSION_RELEASE_UNKNOWN, \
  DDS_PRODUCTVERSION_REVISION_UNKNOWN \
}
```

The value used when the product version is unknown.

7.124.3 Typedef Documentation

7.124.3.1 DDS_Locator_t

```
typedef struct  DDS_Locator_t  DDS_Locator_t
```

<<***extension***>> *(*p. *213)* Type used to represent the addressing information needed to send a message to an RTPS Endpoint using one of the supported transports.

7.124.3.2 DDS_ProtocolVersion_t

```
typedef struct  DDS_ProtocolVersion_t  DDS_ProtocolVersion_t
```

<<***extension***>> *(*p. *432)* Type used to represent the version of the RTPS protocol.

7.124.3.3 DDS_BuiltinTopicKey_t

```
typedef struct  DDS_BuiltinTopicKey_t  DDS_BuiltinTopicKey_t
```

The key type of the built-in topic types.

Each remote **DDSEntity** (p. 1731) to be discovered can be uniquely identified by this key. This is the key of all the built-in topic data types.

See also

DDS_ParticipantBuiltinTopicData (p. 1229)

DDS_TopicBuiltinTopicData (p. 1391)

DDS_PublicationBuiltinTopicData (p. 1264)

DDS_SubscriptionBuiltinTopicData (p. 1370)

7.124.4 Function Documentation

7.124.4.1 DDS_BuiltinTopicKey_equals()

```
DDS_Boolean DDS_BuiltinTopicKey_equals (
            const  DDS_BuiltinTopicKey_t * self,
            const  DDS_BuiltinTopicKey_t * other )
```

Compares this Key with another Key for equality.

Parameters

self	<<***in***>> *(p. 433)* This Key. Cannot be NULL.
other	<<***in***>> *(p. 433)* The other Key to be compared with this Key. Cannot be NULL.

Returns

DDS_BOOLEAN_TRUE (p. 513) if the two Keys have equal values, or **DDS_BOOLEAN_FALSE** (p. 513) otherwise.

7.124.4.2 DDS_BuiltinTopicKey_copy()

```
void DDS_BuiltinTopicKey_copy (
             DDS_BuiltinTopicKey_t * dst,
            const  DDS_BuiltinTopicKey_t * src )
```

Copies another Key into this Key.

Parameters

dst	<<***out***>> *(*p. *433)* This Key. Cannot be NULL.
src	<<***in***>> *(*p. *433)* The other Key to be copied. Cannot be NULL.

7.124.4.3 DDS_BuiltinTopicKey_to_guid()

```
void DDS_BuiltinTopicKey_to_guid (
             const  DDS_BuiltinTopicKey_t * self,
              DDS_GUID_t * dst )
```

Converts this Key into a GUID.

Parameters

self	<<***in***>> *(*p. *433)* This Key. Cannot be NULL.
dst	<<***out***>> *(*p. *433)* The destination GUID. Cannot be NULL.

7.124.4.4 DDS_BuiltinTopicKey_from_guid()

```
void DDS_BuiltinTopicKey_from_guid (
              DDS_BuiltinTopicKey_t * self,
             const  DDS_GUID_t * src )
```

Initializes this Key from the input GUID.

Parameters

self	<<***out***>> *(*p. *433)* This Key. Cannot be NULL.
src	<<***in***>> *(*p. *433)* The GUID to be used to initialize this Key. Cannot be NULL.

7.124.4.5 DDS_BuiltinTopicKey_to_instance_handle()

```
DDS_Boolean DDS_BuiltinTopicKey_to_instance_handle (
             const  DDS_BuiltinTopicKey_t * self,
              DDS_InstanceHandle_t * dst )
```

Converts this Key into an InstanceHandle.

Parameters

self	<<***in***>> *(p. 433)* This Key. Cannot be NULL.
dst	<<***out***>> *(p. 433)* The destination InstanceHandle. Cannot be NULL.

Returns

DDS_BOOLEAN_TRUE (p. 513) on success or **DDS_BOOLEAN_FALSE** (p. 513) on failure

7.124.4.6 DDS_BuiltinTopicKey_from_instance_handle()

```
DDS_Boolean DDS_BuiltinTopicKey_from_instance_handle (
            DDS_BuiltinTopicKey_t * self,
           const  DDS_InstanceHandle_t * src )
```

Initializes this Key from the input InstanceHandle.

Parameters

self	<<***out***>> *(p. 433)* This Key. Cannot be NULL.
src	<<***in***>> *(p. 433)* The InstanceHandle to be used to initialize this Key. Cannot be NULL.

Returns

DDS_BOOLEAN_TRUE (p. 513) on success or **DDS_BOOLEAN_FALSE** (p. 513) on failure

7.124.5 Variable Documentation

7.124.5.1 DDS_LOCATOR_INVALID

```
const struct  DDS_Locator_t DDS_LOCATOR_INVALID  [extern]
```

An invalid locator.

7.124.5.2 DDS_LOCATOR_KIND_INVALID

```
const  DDS_Long DDS_LOCATOR_KIND_INVALID  [extern]
```

Locator of this kind is invalid.

7.124.5.3 DDS_LOCATOR_PORT_INVALID

```
const DDS_UnsignedLong DDS_LOCATOR_PORT_INVALID [extern]
```

An invalid port.

7.124.5.4 DDS_LOCATOR_ADDRESS_INVALID

```
const DDS_Octet DDS_LOCATOR_ADDRESS_INVALID[ DDS_LOCATOR_ADDRESS_LENGTH_MAX] [extern]
```

An invalid address.

7.124.5.5 DDS_LOCATOR_KIND_UDPv4

```
const DDS_Long DDS_LOCATOR_KIND_UDPv4 [extern]
```

A locator for a UDPv4 address.

7.124.5.6 DDS_LOCATOR_KIND_UDPv4_WAN

```
const DDS_Long DDS_LOCATOR_KIND_UDPv4_WAN [extern]
```

A locator for a UDPv4 asymmetric transport address.

7.124.5.7 DDS_LOCATOR_KIND_SHMEM

```
const DDS_Long DDS_LOCATOR_KIND_SHMEM [extern]
```

A locator for an address acessed via shared memory.

7.124.5.8 DDS_LOCATOR_KIND_SHMEM_510

```
const DDS_Long DDS_LOCATOR_KIND_SHMEM_510 [extern]
```

A locator for an address acessed via shared memory for RTI Connext 5.1.0 and earlier.

7.124.5.9 DDS_LOCATOR_KIND_UDPv6

```
const DDS_Long DDS_LOCATOR_KIND_UDPv6 [extern]
```

A locator for a UDPv6 address.

7.124.5.10 DDS_LOCATOR_KIND_UDPv6_510

```
const DDS_Long DDS_LOCATOR_KIND_UDPv6_510 [extern]
```

A locator for a UDPv6 address for RTI Connext 5.1.0 and earlier.

7.124.5.11 DDS_LOCATOR_KIND_RESERVED

```
const DDS_Long DDS_LOCATOR_KIND_RESERVED [extern]
```

Locator of this kind is reserved.

7.124.5.12 value

```
DDS_BUILTIN_TOPIC_KEY_TYPE_NATIVE DDS_BuiltinTopicKey_t::value[DDS_BUILTIN_TOPIC_KEY_TYPE_NATIVE↩
_LENGTH]
```

An array of four integers that uniquely represents a remote **DDSEntity** (p. 1731).

7.124.5.13 DDS_BUILTIN_TOPIC_KEY_UNKNOWN

```
const struct DDS_BuiltinTopicKey_t DDS_BUILTIN_TOPIC_KEY_UNKNOWN [extern]
```

Constant representing an unknown value.

7.125 String Built-in Type

Built-in type consisting of a single character string.

Classes

- class **DDSStringTypeSupport**
  *<<**interface**>> (p. 433) String type support.*
- class **DDSStringDataReader**
  *<<**interface**>> (p. 433) Instantiates `DataReader` < char* >.*
- class **DDSStringDataWriter**
  *<<**interface**>> (p. 433) Instantiates `DataWriter` < char* >.*

7.125.1 Detailed Description

Built-in type consisting of a single character string.

7.126 KeyedString Built-in Type

Built-in type consisting of a string payload and a second string that is the key.

Classes

- struct **DDS_KeyedString**
  Keyed string built-in type.
- struct **DDS_KeyedStringSeq**
  *Instantiates **`FooSeq`** (p. 1965) < **DDS_KeyedString** (p. 1171) > .*
- class **DDSKeyedStringTypeSupport**
  *<<**interface**>> (p. 433) Keyed string type support.*
- class **DDSKeyedStringDataReader**
  *<<**interface**>> (p. 433) Instantiates `DataReader` < **DDS_KeyedString** (p. 1171) >.*
- class **DDSKeyedStringDataWriter**
  *<<**interface**>> (p. 433) Instantiates `DataWriter` < **DDS_KeyedString** (p. 1171) >.*

Typedefs

- typedef struct **DDS_KeyedString** **DDS_KeyedString**
  Keyed string built-in type.

7.126.1 Detailed Description

Built-in type consisting of a string payload and a second string that is the key.

7.126.2 Typedef Documentation

7.126.2.1 DDS_KeyedString

```
typedef struct DDS_KeyedString DDS_KeyedString
```

Keyed string built-in type.

7.127 Octets Built-in Type

Built-in type consisting of a variable-length array of opaque bytes.

Classes

- struct **DDS_Octets**

  Built-in type consisting of a variable-length array of opaque bytes.
- struct **DDS_OctetsSeq**

  Instantiates ***FooSeq*** *(p. 1965) < **DDS_Octets** (p. 1217) > .*
- class **DDSOctetsTypeSupport**

  *<<**interface**>> (p. 433) **DDS_Octets** (p. 1217) type support.*
- class **DDSOctetsDataReader**

  *<<**interface**>> (p. 433) Instantiates `DataReader` < **DDS_Octets** (p. 1217) >.*
- class **DDSOctetsDataWriter**

  *<<**interface**>> (p. 433) Instantiates `DataWriter` < **DDS_Octets** (p. 1217) >.*

Typedefs

- typedef struct **DDS_Octets** **DDS_Octets**

  Built-in type consisting of a variable-length array of opaque bytes.

7.127.1 Detailed Description

Built-in type consisting of a variable-length array of opaque bytes.

7.127.2 Typedef Documentation

7.127.2.1 DDS_Octets

```
typedef struct DDS_Octets DDS_Octets
```

Built-in type consisting of a variable-length array of opaque bytes.

7.128 KeyedOctets Built-in Type

Built-in type consisting of a variable-length array of opaque bytes and a string that is the key.

Classes

- struct **DDS_KeyedOctets**

  Built-in type consisting of a variable-length array of opaque bytes and a string that is the key.
- struct **DDS_KeyedOctetsSeq**

  Instantiates ***FooSeq*** *(p. 1965) < **DDS_KeyedOctets** (p. 1168) >.*
- class **DDSKeyedOctetsTypeSupport**

  *<<**interface**>> (p. 433) **DDS_KeyedOctets** (p. 1168) type support.*
- class **DDSKeyedOctetsDataReader**

  *<<**interface**>> (p. 433) Instantiates `DataReader` < **DDS_KeyedOctets** (p. 1168) >.*
- class **DDSKeyedOctetsDataWriter**

  *<<**interface**>> (p. 433) Instantiates `DataWriter` < **DDS_KeyedOctets** (p. 1168) >.*

Typedefs

- typedef struct **DDS_KeyedOctets** **DDS_KeyedOctets**

  Built-in type consisting of a variable-length array of opaque bytes and a string that is the key.

7.128.1 Detailed Description

Built-in type consisting of a variable-length array of opaque bytes and a string that is the key.

7.128.2 Typedef Documentation

7.128.2.1 DDS_KeyedOctets

`typedef struct` **`DDS_KeyedOctets`** **`DDS_KeyedOctets`**

Built-in type consisting of a variable-length array of opaque bytes and a string that is the key.

7.129 DDS-Specific Primitive Types

Basic DDS value types for use in user data types.

Macros

- #define **DDS_BOOLEAN_TRUE**

  *Defines "true" value of **DDS_Boolean** (p. 516) data type.*
- #define **DDS_BOOLEAN_FALSE**

  *Defines "false" value of **DDS_Boolean** (p. 516) data type.*

Typedefs

- typedef RTICdrChar **DDS_Char**

  Defines a character data type, equivalent to IDL/CDR `char`.
- typedef RTICdrWchar **DDS_Wchar**

  Defines a wide character data type, equivalent to IDL/CDR `wchar`.
- typedef RTICdrOctet **DDS_Octet**

  Defines an opaque byte data type, equivalent to IDL/CDR `octet`.
- typedef RTICdrOctet **DDS_UInt8**

  Defines a data type, equivalent to IDL/CDR `uint8`.
- typedef RTICdrInt8 **DDS_Int8**

  Defines a data type, equivalent to IDL/CDR `int8`.
- typedef RTICdrShort **DDS_Short**

  Defines a short integer data type, equivalent to IDL/CDR `short`.
- typedef RTICdrUnsignedShort **DDS_UnsignedShort**

  Defines an unsigned short integer data type, equivalent to IDL/CDR `unsigned short`.
- typedef RTICdrLong **DDS_Long**

  Defines a long integer data type, equivalent to IDL/CDR `long`.
- typedef RTICdrUnsignedLong **DDS_UnsignedLong**

  Defines an unsigned long integer data type, equivalent to IDL/CDR `unsigned long`.
- typedef RTICdrLongLong **DDS_LongLong**

  Defines an extra-long integer data type, equivalent to IDL/CDR `long long`.
- typedef RTICdrUnsignedLongLong **DDS_UnsignedLongLong**

  Defines an unsigned extra-long data type, equivalent to IDL/CDR `unsigned long long`.
- typedef RTICdrFloat **DDS_Float**

  Defines a single-precision floating-point data type, equivalent to IDL/CDR `float`.
- typedef RTICdrDouble **DDS_Double**

  Defines a double-precision floating-point data type, equivalent to IDL/CDR `double`.
- typedef RTICdrLongDouble **DDS_LongDouble**

  Defines an extra-precision floating-point data type, equivalent to IDL/CDR `long double`.
- typedef RTICdrBoolean **DDS_Boolean**

  Defines a Boolean data type, equivalent to IDL/CDR `boolean`.
- typedef RTICdrEnum **DDS_Enum**

  Defines an enumerated data type.

7.129.1 Detailed Description

Basic DDS value types for use in user data types.

As part of the finalization of the DDS standard, a number of DDS-specific primitive types will be introduced. By using these types, you will ensure that your data is serialized consistently across platforms even if the C/C++ built-in types have different sizes on those platforms.

In this version of RTI Connext, the DDS primitive types are defined using the OMG's Common Data Representation (CDR) standard. In a future version of RTI Connext, you will be given the choice of whether to use these CDR-based types or C/C++ built-in types through a flag provided to the rtiddsgen tool.

Typedef's that begin with RTICdr are defined in <NDDSHOME>/include/ndds/cdr/cdr_type.h, which uses types that are further defined in <NDDSHOME>/include/ndds/osapi/osapi_type.h.

7.129.2 Macro Definition Documentation

7.129.2.1 DDS_BOOLEAN_TRUE

```
#define DDS_BOOLEAN_TRUE
```

Defines "true" value of **DDS_Boolean** (p. 516) data type.

Examples

HelloWorld.cxx.

7.129.2.2 DDS_BOOLEAN_FALSE

```
#define DDS_BOOLEAN_FALSE
```

Defines "false" value of **DDS_Boolean** (p. 516) data type.

Examples

HelloWorld.cxx.

7.129.3 Typedef Documentation

7.129.3.1 DDS_Char

```
typedef RTICdrChar DDS_Char
```

Defines a character data type, equivalent to IDL/CDR `char`.

An 8-bit quantity that encodes a single byte charater from any byte-oriented code set.

7.129.3.2 DDS_Wchar

```
typedef RTICdrWchar DDS_Wchar
```

Defines a wide character data type, equivalent to IDL/CDR `wchar`.

A 16-bit quantity that contains a wide character encoded in UTF-16.

7.129.3.3 DDS_Octet

```
typedef RTICdrOctet DDS_Octet
```

Defines an opaque byte data type, equivalent to IDL/CDR `octet`.

An 8-bit quantity that is guaranteed not to undergo any conversion when transmitted by the middleware.

7.129.3.4 DDS_UInt8

```
typedef RTICdrOctet DDS_UInt8
```

Defines a data type, equivalent to IDL/CDR `uint8`.

An 8-bit quantity that is guaranteed not to undergo any conversion when transmitted by the middleware.

7.129.3.5 DDS_Int8

```
typedef RTICdrInt8 DDS_Int8
```

Defines a data type, equivalent to IDL/CDR `int8`.

An 8-bit quantity that is guaranteed not to undergo any conversion when transmitted by the middleware.

7.129.3.6 DDS_Short

```
typedef RTICdrShort DDS_Short
```

Defines a short integer data type, equivalent to IDL/CDR `short`.

A 16-bit signed short integer value.

7.129.3.7 DDS_UnsignedShort

```
typedef RTICdrUnsignedShort DDS_UnsignedShort
```

Defines an unsigned short integer data type, equivalent to IDL/CDR `unsigned short`.

A 16-bit unsigned short integer value.

7.129.3.8 DDS_Long

```
typedef RTICdrLong DDS_Long
```

Defines a long integer data type, equivalent to IDL/CDR `long`.

A 32-bit signed long integer value.

7.129.3.9 DDS_UnsignedLong

```
typedef RTICdrUnsignedLong DDS_UnsignedLong
```

Defines an unsigned long integer data type, equivalent to IDL/CDR `unsigned long`.

A 32-bit unsigned long integer value.

7.129.3.10 DDS_LongLong

```
typedef RTICdrLongLong DDS_LongLong
```

Defines an extra-long integer data type, equivalent to IDL/CDR `long long`.

A 64-bit signed long long integer value.

7.129.3.11 DDS_UnsignedLongLong

```
typedef RTICdrUnsignedLongLong DDS_UnsignedLongLong
```

Defines an unsigned extra-long data type, equivalent to IDL/CDR `unsigned long long`.

A 64-bit unsigned long long integer value.

7.129.3.12 DDS_Float

```
typedef RTICdrFloat DDS_Float
```

Defines a single-precision floating-point data type, equivalent to IDL/CDR `float`.

A 32-bit floating-point value.

7.129.3.13 DDS_Double

```
typedef RTICdrDouble DDS_Double
```

Defines a double-precision floating-point data type, equivalent to IDL/CDR `double`.

A 64-bit floating-point value.

7.129.3.14 DDS_LongDouble

```
typedef RTICdrLongDouble DDS_LongDouble
```

Defines an extra-precision floating-point data type, equivalent to IDL/CDR `long double`.

A 128-bit floating-point value.

Since some architectures do not support long double, RTI has defined character arrays that match the expected size of this type. On systems that do have native long double, you have to define RTI_CDR_SIZEOF_LONG_DOUBLE as 16 to map them to native types.

7.129.3.15 DDS_Boolean

```
typedef RTICdrBoolean DDS_Boolean
```

Defines a Boolean data type, equivalent to IDL/CDR `boolean`.

An 8-bit Boolean value that is used to denote a data item that can only take one of the values **DDS_BOOLEAN_TRUE** (p. 513) (1) or **DDS_BOOLEAN_FALSE** (p. 513) (0).

Examples

HelloWorld.cxx.

7.129.3.16 DDS_Enum

```
typedef RTICdrEnum DDS_Enum
```

Defines an enumerated data type.

Encoded as a signed 32-bit integer value. By default, the first enum identifier has the numeric value zero (0) (unless the value is provided explicitly). Successive enum identifiers take ascending numeric values, in order of declaration from left to right.

7.130 Time Support

Time and duration types and defines.

Classes

- struct **DDS_Duration_t**

  Type for duration representation.
- struct **DDS_Time_t**

  Type for time representation.

Macros

- #define **DDS_TIME_ZERO**

  The default instant in time: zero seconds and zero nanoseconds.

Functions

- static **DDS_Duration_t** **DDS_Duration_t::from_micros** (**DDS_UnsignedLongLong** microseconds)

  Creates a new duration object from a duration expressed in microseconds.
- static **DDS_Duration_t** **DDS_Duration_t::from_millis** (**DDS_UnsignedLongLong** milliseconds)

  Creates a new duration object from a duration expressed in milliseconds.
- static **DDS_Duration_t** **DDS_Duration_t::from_nanos** (**DDS_UnsignedLongLong** nanoseconds)

  Creates a new duration object from a duration expressed in nanoseconds.
- static **DDS_Duration_t** **DDS_Duration_t::from_seconds** (**DDS_UnsignedLong** seconds)

  Creates a new duration object from a duration expressed in seconds.
- static **DDS_Time_t** **DDS_Time_t::from_micros** (**DDS_UnsignedLongLong** microseconds)

  Creates a new time object from a time expressed in microseconds.
- static **DDS_Time_t** **DDS_Time_t::from_millis** (**DDS_UnsignedLongLong** milliseconds)

  Creates a new time object from a time expressed in milliseconds.
- static **DDS_Time_t** **DDS_Time_t::from_nanos** (**DDS_UnsignedLongLong** nanoseconds)

  Creates a new time object from a time expressed in nanoseconds.
- static **DDS_Time_t** **DDS_Time_t::from_seconds** (**DDS_UnsignedLongLong** seconds)

  Creates a new time object from a time expressed in seconds.
- **DDS_Boolean** **DDS_Time_is_zero** (const struct **DDS_Time_t** *time)

  Check if time is zero.
- **DDS_Boolean** **DDS_Time_is_invalid** (const struct **DDS_Time_t** *time)
- **DDS_Boolean** **DDS_Duration_is_infinite** (const struct **DDS_Duration_t** *duration)
- **DDS_Boolean** **DDS_Duration_is_auto** (const struct **DDS_Duration_t** *duration)
- **DDS_Boolean** **DDS_Duration_is_zero** (const struct **DDS_Duration_t** *duration)

Variables

- const struct **DDS_Time_t DDS_TIME_MAX**

  The maximum value of time.
- const **DDS_LongLong DDS_TIME_INVALID_SEC**

  A sentinel indicating an invalid second of time.
- const **DDS_UnsignedLong DDS_TIME_INVALID_NSEC**

  A sentinel indicating an invalid nano-second of time.
- const struct **DDS_Time_t DDS_TIME_INVALID**

  A sentinel indicating an invalid time.
- const **DDS_Long DDS_DURATION_INFINITE_SEC**

  An infinite second period of time.
- const **DDS_UnsignedLong DDS_DURATION_INFINITE_NSEC**

  An infinite nano-second period of time.
- const struct **DDS_Duration_t DDS_DURATION_INFINITE**

  An infinite period of time.
- const **DDS_Long DDS_DURATION_AUTO_SEC**

  An auto second period of time.
- const **DDS_UnsignedLong DDS_DURATION_AUTO_NSEC**

  An auto nano-second period of time.
- const struct **DDS_Duration_t DDS_DURATION_AUTO**

  Duration is automatically assigned.
- const **DDS_Long DDS_DURATION_ZERO_SEC**

  A zero-length second period of time.
- const **DDS_UnsignedLong DDS_DURATION_ZERO_NSEC**

  A zero-length nano-second period of time.
- const struct **DDS_Duration_t DDS_DURATION_ZERO**

  A zero-length period of time.

7.130.1 Detailed Description

Time and duration types and defines.

7.130.2 Macro Definition Documentation

7.130.2.1 DDS_TIME_ZERO

```
#define DDS_TIME_ZERO
```

The default instant in time: zero seconds and zero nanoseconds.

7.130.3 Function Documentation

7.130.3.1 from_micros() [1/2]

```
static DDS_Duration_t DDS_Duration_t::from_micros (
            DDS_UnsignedLongLong microseconds ) [static]
```

Creates a new duration object from a duration expressed in microseconds.

In case of an overflow this function returns **DDS_DURATION_INFINITE** (p. 522).

7.130.3.2 from_millis() [1/2]

```
static DDS_Duration_t DDS_Duration_t::from_millis (
            DDS_UnsignedLongLong milliseconds ) [static]
```

Creates a new duration object from a duration expressed in milliseconds.

In case of an overflow this function returns **DDS_DURATION_INFINITE** (p. 522).

7.130.3.3 from_nanos() [1/2]

```
static DDS_Duration_t DDS_Duration_t::from_nanos (
            DDS_UnsignedLongLong nanoseconds ) [static]
```

Creates a new duration object from a duration expressed in nanoseconds.

In case of an overflow this function returns **DDS_DURATION_INFINITE** (p. 522).

7.130.3.4 from_seconds() [1/2]

```
static DDS_Duration_t DDS_Duration_t::from_seconds (
            DDS_UnsignedLong seconds ) [static]
```

Creates a new duration object from a duration expressed in seconds.

In case of an overflow this function returns **DDS_DURATION_INFINITE** (p. 522).

7.130.3.5 from_micros() [2/2]

```
static DDS_Time_t DDS_Time_t::from_micros (
            DDS_UnsignedLongLong microseconds ) [static]
```

Creates a new time object from a time expressed in microseconds.

In case of an overflow this function returns **DDS_TIME_MAX** (p. 521).

7.130.3.6 from_millis() [2/2]

```
static DDS_Time_t DDS_Time_t::from_millis (
            DDS_UnsignedLongLong milliseconds ) [static]
```

Creates a new time object from a time expressed in milliseconds.

In case of an overflow this function returns **DDS_TIME_MAX** (p. 521).

7.130.3.7 from_nanos() [2/2]

```
static DDS_Time_t DDS_Time_t::from_nanos (
            DDS_UnsignedLongLong nanoseconds ) [static]
```

Creates a new time object from a time expressed in nanoseconds.

In case of an overflow this function returns **DDS_TIME_MAX** (p. 521).

7.130.3.8 from_seconds() [2/2]

```
static DDS_Time_t DDS_Time_t::from_seconds (
            DDS_UnsignedLongLong seconds ) [static]
```

Creates a new time object from a time expressed in seconds.

In case of an overflow this function returns **DDS_TIME_MAX** (p. 521).

7.130.3.9 DDS_Time_is_zero()

```
DDS_Boolean DDS_Time_is_zero (
            const struct DDS_Time_t * time )
```

Check if time is zero.

Returns

DDS_BOOLEAN_TRUE (p. 513) if the given time is equal to **DDS_TIME_ZERO** (p. 518) or **DDS_BOOLEAN_↩ FALSE** (p. 513) otherwise.

7.130.3.10 DDS_Time_is_invalid()

```
DDS_Boolean DDS_Time_is_invalid (
            const struct DDS_Time_t * time )
```

Returns

DDS_BOOLEAN_TRUE (p. 513) if the given time is not valid (i.e. is negative)

7.130.3.11 DDS_Duration_is_infinite()

```
DDS_Boolean DDS_Duration_is_infinite (
            const struct  DDS_Duration_t * duration )
```

Returns

DDS_BOOLEAN_TRUE (p. 513) if the given duration is of infinite length.

7.130.3.12 DDS_Duration_is_auto()

```
DDS_Boolean DDS_Duration_is_auto (
            const struct  DDS_Duration_t * duration )
```

Returns

DDS_BOOLEAN_TRUE (p. 513) if the given duration has auto value.

7.130.3.13 DDS_Duration_is_zero()

```
DDS_Boolean DDS_Duration_is_zero (
            const struct  DDS_Duration_t * duration )
```

Returns

DDS_BOOLEAN_TRUE (p. 513) if the given duration is of zero length.

7.130.4 Variable Documentation

7.130.4.1 DDS_TIME_MAX

```
const struct  DDS_Time_t DDS_TIME_MAX  [extern]
```

The maximum value of time.

7.130.4.2 DDS_TIME_INVALID_SEC

```
const DDS_LongLong DDS_TIME_INVALID_SEC [extern]
```

A sentinel indicating an invalid second of time.

7.130.4.3 DDS_TIME_INVALID_NSEC

```
const DDS_UnsignedLong DDS_TIME_INVALID_NSEC [extern]
```

A sentinel indicating an invalid nano-second of time.

7.130.4.4 DDS_TIME_INVALID

```
const struct DDS_Time_t DDS_TIME_INVALID [extern]
```

A sentinel indicating an invalid time.

7.130.4.5 DDS_DURATION_INFINITE_SEC

```
const DDS_Long DDS_DURATION_INFINITE_SEC [extern]
```

An infinite second period of time.

7.130.4.6 DDS_DURATION_INFINITE_NSEC

```
const DDS_UnsignedLong DDS_DURATION_INFINITE_NSEC [extern]
```

An infinite nano-second period of time.

7.130.4.7 DDS_DURATION_INFINITE

```
const struct DDS_Duration_t DDS_DURATION_INFINITE [extern]
```

An infinite period of time.

7.130.4.8 DDS_DURATION_AUTO_SEC

```
const DDS_Long DDS_DURATION_AUTO_SEC [extern]
```

An auto second period of time.

7.130.4.9 DDS_DURATION_AUTO_NSEC

```
const DDS_UnsignedLong DDS_DURATION_AUTO_NSEC [extern]
```

An auto nano-second period of time.

7.130.4.10 DDS_DURATION_AUTO

```
const struct DDS_Duration_t DDS_DURATION_AUTO [extern]
```

Duration is automatically assigned.

7.130.4.11 DDS_DURATION_ZERO_SEC

```
const DDS_Long DDS_DURATION_ZERO_SEC [extern]
```

A zero-length second period of time.

7.130.4.12 DDS_DURATION_ZERO_NSEC

```
const DDS_UnsignedLong DDS_DURATION_ZERO_NSEC [extern]
```

A zero-length nano-second period of time.

7.130.4.13 DDS_DURATION_ZERO

```
const struct DDS_Duration_t DDS_DURATION_ZERO [extern]
```

A zero-length period of time.

7.131 GUID Support

<<***extension***>> *(p. 432)* GUID type and defines.

Classes

- struct **DDS_RTPS_EntityId_t**

  From the DDS-RTPS specification: type used to hold the suffix part of the globally-unique RTPS-entity identifiers.
- struct **DDS_RTPS_GUID_t**

  From the DDS-RTPS specification: type used to hold a globally-unique RTPS-entity identifier.
- struct **DDS_GUID_t**

  Type for GUID (Global Unique Identifier) representation.

Typedefs

- typedef **DDS_Octet** **DDS_RTPS_GuidPrefix_t**[DDS_RTPS_GUID_PREFIX_LENGTH]

  From the DDS-RTPS specification: type used to hold the prefix of the globally-unique RTPS-entity identifiers.
- typedef struct **DDS_RTPS_EntityId_t** **DDS_RTPS_EntityId_t**

  From the DDS-RTPS specification: type used to hold the suffix part of the globally-unique RTPS-entity identifiers.
- typedef struct **DDS_RTPS_GUID_t** **DDS_RTPS_GUID_t**

  From the DDS-RTPS specification: type used to hold a globally-unique RTPS-entity identifier.
- typedef struct **DDS_GUID_t** **DDS_GUID_t**

  Type for GUID (Global Unique Identifier) representation.

Functions

- **DDS_Boolean** **DDS_GUID_equals** (const struct **DDS_GUID_t** *self, const struct **DDS_GUID_t** *other)

  Compares this GUID with another GUID for equality.
- int **DDS_GUID_compare** (const struct **DDS_GUID_t** *self, const struct **DDS_GUID_t** *other)

  Compares two GUIDs.
- void **DDS_GUID_copy** (struct **DDS_GUID_t** *self, const struct **DDS_GUID_t** *other)

  Copies another GUID into this GUID.

Variables

- const struct **DDS_GUID_t** **DDS_GUID_AUTO**

  Indicates that RTI Connext should choose an appropriate virtual GUID.
- const struct **DDS_GUID_t** **DDS_GUID_UNKNOWN**

  Unknown GUID.
- const struct **DDS_GUID_t** **DDS_GUID_ZERO**

  Zero GUID.

7.131.1 Detailed Description

<<***extension***>> *(p. 432)* GUID type and defines.

7.131.2 Typedef Documentation

7.131.2.1 DDS_RTPS_GuidPrefix_t

```
typedef  DDS_Octet DDS_RTPS_GuidPrefix_t[DDS_RTPS_GUID_PREFIX_LENGTH]
```

From the DDS-RTPS specification: type used to hold the prefix of the globally-unique RTPS-entity identifiers.

From DDS-RTPS Specification, clauses 8.4.2.1 and 9.3.1.

7.131.2.2 DDS_RTPS_EntityId_t

```
typedef struct  DDS_RTPS_EntityId_t  DDS_RTPS_EntityId_t
```

From the DDS-RTPS specification: type used to hold the suffix part of the globally-unique RTPS-entity identifiers.

From DDS-RTPS Specification, clauses 8.4.2.1 and 9.3.1.

7.131.2.3 DDS_RTPS_GUID_t

```
typedef struct  DDS_RTPS_GUID_t  DDS_RTPS_GUID_t
```

From the DDS-RTPS specification: type used to hold a globally-unique RTPS-entity identifier.

From DDS-RTPS Specification, clauses 8.4.2.1 and 9.3.1.

7.131.2.4 DDS_GUID_t

```
typedef struct  DDS_GUID_t  DDS_GUID_t
```

Type for *GUID* (Global Unique Identifier) representation.

Represents a 128 bit GUID.

Alternative representation of **DDS_RTPS_GUID_t** (p. 1314). Memory and wire representation for this type is the same as the one for **DDS_RTPS_GUID_t** (p. 1314).

7.131.3 Function Documentation

7.131.3.1 DDS_GUID_equals()

```
DDS_Boolean DDS_GUID_equals (
            const struct  DDS_GUID_t * self,
            const struct  DDS_GUID_t * other )
```

Compares this GUID with another GUID for equality.

Parameters

self	<<***in***>> *(p. 433)* This GUID.
other	<<***in***>> *(p. 433)* The other GUID to be compared with this GUID.

Returns

DDS_BOOLEAN_TRUE (p. 513) if the two GUIDs have equal values, or **DDS_BOOLEAN_FALSE** (p. 513) otherwise.

7.131.3.2 DDS_GUID_compare()

```
int DDS_GUID_compare (
            const struct  DDS_GUID_t * self,
            const struct  DDS_GUID_t * other )
```

Compares two GUIDs.

Parameters

self	<<***in***>> *(p. 433)* GUID to compare.
other	<<***in***>> *(p. 433)* GUID to compare.

Returns

If the two GUIDs are equal or NULL, the function returns 0. If self is greater than other, the function returns a positive number; otherwise, it returns a negative number. Note that non-NULL is considered greater than NULL.

7.131.3.3 DDS_GUID_copy()

```
void DDS_GUID_copy (
            struct  DDS_GUID_t * self,
            const struct  DDS_GUID_t * other )
```

Copies another GUID into this GUID.

Parameters

self	<<***in***>> *(p. 433)* This GUID. Cannot be NULL.
other	<<***in***>> *(p. 433)* The other GUID to be copied. Cannot be NULL.

7.131.4 Variable Documentation

7.131.4.1 DDS_GUID_AUTO

```
const struct DDS_GUID_t DDS_GUID_AUTO [extern]
```

Indicates that RTI Connext should choose an appropriate virtual GUID.

If this special value is assigned to **DDS_DataWriterProtocolQosPolicy::virtual_guid** (p. 893) or **DDS_DataReader↩ ProtocolQosPolicy::virtual_guid** (p. 844), RTI Connext will assign the virtual GUID automatically based on the RTPS or physical GUID.

7.131.4.2 DDS_GUID_UNKNOWN

```
const struct DDS_GUID_t DDS_GUID_UNKNOWN [extern]
```

Unknown GUID.

7.131.4.3 DDS_GUID_ZERO

```
const struct DDS_GUID_t DDS_GUID_ZERO [extern]
```

Zero GUID.

7.132 Sequence Number Support

<<***extension***>> *(p. 432)* Sequence number type and defines.

Classes

- struct **DDS_SequenceNumber_t**
  Type for sequence number representation.

Typedefs

- typedef struct **DDS_SequenceNumber_t** **DDS_SequenceNumber_t**
  Type for sequence number representation.

Variables

- const struct **DDS_SequenceNumber_t DDS_SEQUENCE_NUMBER_UNKNOWN**
  Unknown sequence number.
- const struct **DDS_SequenceNumber_t DDS_SEQUENCE_NUMBER_ZERO**
  Zero value for the sequence number.
- const struct **DDS_SequenceNumber_t DDS_SEQUENCE_NUMBER_MAX**
  Highest, most positive value for the sequence number.
- const struct **DDS_SequenceNumber_t DDS_AUTO_SEQUENCE_NUMBER**
  The sequence number is internally determined by RTI Connext.

7.132.1 Detailed Description

<<***extension***>> *(*p. *432)* Sequence number type and defines.

7.132.2 Typedef Documentation

7.132.2.1 DDS_SequenceNumber_t

```
typedef struct DDS_SequenceNumber_t DDS_SequenceNumber_t
```

Type for *sequence* number representation.

Represents a 64-bit sequence number.

7.132.3 Variable Documentation

7.132.3.1 DDS_SEQUENCE_NUMBER_UNKNOWN

```
const struct DDS_SequenceNumber_t DDS_SEQUENCE_NUMBER_UNKNOWN [extern]
```

Unknown sequence number.

7.132.3.2 DDS_SEQUENCE_NUMBER_ZERO

```
const struct DDS_SequenceNumber_t DDS_SEQUENCE_NUMBER_ZERO [extern]
```

Zero value for the sequence number.

7.132.3.3 DDS_SEQUENCE_NUMBER_MAX

```
const struct DDS_SequenceNumber_t DDS_SEQUENCE_NUMBER_MAX [extern]
```

Highest, most positive value for the sequence number.

7.132.3.4 DDS_AUTO_SEQUENCE_NUMBER

```
const struct DDS_SequenceNumber_t DDS_AUTO_SEQUENCE_NUMBER [extern]
```

The sequence number is internally determined by RTI Connext.

7.133 Exception Codes

<<***extension***>> *(*p. *432)* Exception codes.

Enumerations

- enum **DDS_ExceptionCode_t** {
  DDS_NO_EXCEPTION_CODE ,
  DDS_USER_EXCEPTION_CODE ,
  DDS_SYSTEM_EXCEPTION_CODE ,
  DDS_BAD_PARAM_SYSTEM_EXCEPTION_CODE ,
  DDS_NO_MEMORY_SYSTEM_EXCEPTION_CODE ,
  DDS_BAD_TYPECODE_SYSTEM_EXCEPTION_CODE ,
  DDS_BADKIND_USER_EXCEPTION_CODE ,
  DDS_BOUNDS_USER_EXCEPTION_CODE ,
  DDS_IMMUTABLE_TYPECODE_SYSTEM_EXCEPTION_CODE = 8 ,
  DDS_BAD_MEMBER_NAME_USER_EXCEPTION_CODE = 9 ,
  DDS_BAD_MEMBER_ID_USER_EXCEPTION_CODE = 10 }

  *Error codes used by the **DDS_TypeCode** (*p. *1428) class.*

7.133.1 Detailed Description

<<***extension***>> *(p. 432)* Exception codes.

These exceptions are used for error handling by the **Type Code Support** (p. 79) API.

7.133.2 Enumeration Type Documentation

7.133.2.1 DDS_ExceptionCode_t

enum **DDS_ExceptionCode_t**

Error codes used by the **DDS_TypeCode** (p. 1428) class.

Exceptions are modeled via a special parameter passed to the operations.

Enumerator

DDS_NO_EXCEPTION_CODE	No failure occurred.
DDS_USER_EXCEPTION_CODE	User exception. This class is based on a similar class in CORBA.
DDS_SYSTEM_EXCEPTION_CODE	System exception. This class is based on a similar class in CORBA.
DDS_BAD_PARAM_SYSTEM_EXCEPTION_CODE	Exception thrown when a parameter passed to a call is considered illegal.
DDS_NO_MEMORY_SYSTEM_EXCEPTION_CODE	Exception thrown when there is not enough memory for a dynamic memory allocation.
DDS_BAD_TYPECODE_SYSTEM_EXCEPTION_↩ CODE	Exception thrown when a malformed type code is found (for example, a type code with an invalid TCKind value).
DDS_BADKIND_USER_EXCEPTION_CODE	The exception BadKind is thrown when an inappropriate operation is invoked on a TypeCode object.
DDS_BOUNDS_USER_EXCEPTION_CODE	A user exception thrown when a parameter is not within the legal bounds.
DDS_IMMUTABLE_TYPECODE_SYSTEM_↩ EXCEPTION_CODE	An attempt was made to modify a **DDS_TypeCode** (p. 1428) that was received from a remote object. The built-in publication and subscription readers provide access to information about the remote **DDSDataWriter** (p. 1589) and **DDSDataReader** (p. 1556) entities in the distributed system. Among other things, the data from these built-in readers contains the **DDS_TypeCode** (p. 1428) for these entities. Modifying this received **DDS_TypeCode** (p. 1428) is not permitted.

Enumerator

DDS_BAD_MEMBER_NAME_USER_EXCEPTION_↩ CODE	The specified **DDS_TypeCode** (p. 1428) member name is invalid. This failure can occur, for example, when querying a field by name when no such name is defined in the type. **See also** **DDS_BAD_MEMBER_ID_USER_EXCEPTION↩ _CODE** (p. 531)
DDS_BAD_MEMBER_ID_USER_EXCEPTION_CODE	The specified **DDS_TypeCode** (p. 1428) member ID is invalid. This failure can occur, for example, when querying a field by ID when no such ID is defined in the type. **See also** **DDS_BAD_MEMBER_NAME_USER_↩ EXCEPTION_CODE** (p. 531)

7.134 Return Codes

Types of return codes.

Enumerations

- enum **DDS_ReturnCode_t** {
  DDS_RETCODE_OK ,
  DDS_RETCODE_ERROR ,
  DDS_RETCODE_UNSUPPORTED ,
  DDS_RETCODE_BAD_PARAMETER ,
  DDS_RETCODE_PRECONDITION_NOT_MET ,
  DDS_RETCODE_OUT_OF_RESOURCES ,
  DDS_RETCODE_NOT_ENABLED ,
  DDS_RETCODE_IMMUTABLE_POLICY ,
  DDS_RETCODE_INCONSISTENT_POLICY ,
  DDS_RETCODE_ALREADY_DELETED ,
  DDS_RETCODE_TIMEOUT ,
  DDS_RETCODE_NO_DATA ,
  DDS_RETCODE_ILLEGAL_OPERATION ,
  DDS_RETCODE_NOT_ALLOWED_BY_SECURITY }

  Type for return codes.

7.134.1 Detailed Description

Types of return codes.

7.134.2 Standard Return Codes

Any operation with return type **DDS_ReturnCode_t** (p. 532) may return **DDS_RETCODE_OK** (p. 532) **DDS_↩ RETCODE_ERROR** (p. 532) or **DDS_RETCODE_ILLEGAL_OPERATION** (p. 533). Any operation that takes one or more input parameters may additionally return **DDS_RETCODE_BAD_PARAMETER** (p. 532). Any operation on an object created from any of the factories may additionally return **DDS_RETCODE_ALREADY_DELETED** (p. 533). Any operation that is stated as optional may additionally return **DDS_RETCODE_UNSUPPORTED** (p. 532).

Thus, the standard return codes are:

- **DDS_RETCODE_ERROR** (p. 532)
- **DDS_RETCODE_ILLEGAL_OPERATION** (p. 533)
- **DDS_RETCODE_ALREADY_DELETED** (p. 533)
- **DDS_RETCODE_BAD_PARAMETER** (p. 532)
- **DDS_RETCODE_UNSUPPORTED** (p. 532)

Operations that may return any of the additional return codes will state so explicitly.

7.134.3 Enumeration Type Documentation

7.134.3.1 DDS_ReturnCode_t

`enum` **`DDS_ReturnCode_t`**

Type for return codes.

Errors are modeled as operation return codes of this type.

Enumerator

DDS_RETCODE_OK	Successful return.
DDS_RETCODE_ERROR	Generic, unspecified error.
DDS_RETCODE_UNSUPPORTED	Unsupported operation. Only returned by operations that are unsupported.
DDS_RETCODE_BAD_PARAMETER	Illegal parameter value. The value of the parameter that is passed in has llegal value. Things that fall into this category include NULL parameters and parameter values that are out of range.
DDS_RETCODE_PRECONDITION_NOT_MET	A pre-condition for the operation was not met. The system is not in the expected state when the function is called, or the parameter itself is not in the expected state when the function is called.

Enumerator

DDS_RETCODE_OUT_OF_RESOURCES	RTI Connext ran out of the resources needed to complete the operation.
DDS_RETCODE_NOT_ENABLED	Operation invoked on a **DDSEntity** (p. 1731) that is not yet enabled.
DDS_RETCODE_IMMUTABLE_POLICY	Application attempted to modify an immutable QoS policy.
DDS_RETCODE_INCONSISTENT_POLICY	Application specified a set of QoS policies that are not consistent with each other.
DDS_RETCODE_ALREADY_DELETED	The object target of this operation has already been deleted.
DDS_RETCODE_TIMEOUT	The operation timed out.
DDS_RETCODE_NO_DATA	Indicates a transient situation where the operation did not return any data but there is no inherent error.
DDS_RETCODE_ILLEGAL_OPERATION	The operation was called under improper circumstances. An operation was invoked on an inappropriate object or at an inappropriate time. This return code is similar to **DDS_RETCODE_PRECONDITION_NOT_MET** (p. 532), except that there is no precondition that could be changed to make the operation succeed.
DDS_RETCODE_NOT_ALLOWED_BY_SECURITY	An operation on the DDS API that fails because the security plugins do not allow it. An operation on the DDS API that fails because the security plugins do not allow it.

7.135 Status Kinds

Kinds of communication status.

Macros

- #define **DDS_STATUS_MASK_NONE**

  No bits are set.
- #define **DDS_STATUS_MASK_ALL**

  All bits are set.

Typedefs

- typedef **DDS_UnsignedLong DDS_StatusMask**

  A bit-mask (list) of concrete status types, i.e. ***DDS_StatusKind*** *(p. 538)[].*

Enumerations

- enum **DDS_StatusKind** {
  DDS_INCONSISTENT_TOPIC_STATUS ,
  DDS_OFFERED_DEADLINE_MISSED_STATUS ,
  DDS_REQUESTED_DEADLINE_MISSED_STATUS ,
  DDS_OFFERED_INCOMPATIBLE_QOS_STATUS ,
  DDS_REQUESTED_INCOMPATIBLE_QOS_STATUS ,
  DDS_SAMPLE_LOST_STATUS ,
  DDS_SAMPLE_REJECTED_STATUS ,
  DDS_DATA_ON_READERS_STATUS ,
  DDS_DATA_AVAILABLE_STATUS ,
  DDS_LIVELINESS_LOST_STATUS ,
  DDS_LIVELINESS_CHANGED_STATUS ,
  DDS_PUBLICATION_MATCHED_STATUS ,
  DDS_SUBSCRIPTION_MATCHED_STATUS ,
  DDS_DATA_WRITER_EMPTY_VIRTUAL_WRITER_STATUS = 0x0001U << 19 ,
  DDS_INVALID_LOCAL_ADVANCE_NOTICE_STATUS ,
  DDS_SERVICE_REQUEST_ACCEPTED_STATUS ,
  DDS_DATA_WRITER_APPLICATION_ACKNOWLEDGMENT_STATUS ,
  DDS_DATA_WRITER_INSTANCE_REPLACED_STATUS ,
  DDS_RELIABLE_WRITER_CACHE_CHANGED_STATUS ,
  DDS_RELIABLE_READER_ACTIVITY_CHANGED_STATUS ,
  DDS_DATA_WRITER_CACHE_STATUS ,
  DDS_DATA_WRITER_PROTOCOL_STATUS ,
  DDS_DATA_READER_CACHE_STATUS ,
  DDS_DATA_READER_PROTOCOL_STATUS ,
  DDS_DATA_WRITER_DESTINATION_UNREACHABLE_STATUS = 0x00000001U << 30 ,
  DDS_DATA_WRITER_SAMPLE_REMOVED_STATUS }

  Type for status kinds.

7.135.1 Detailed Description

Kinds of communication status.

Entity:

DDSEntity (p. 1731)

QoS:

All QoS Policies (p. 549)

Listener:

DDSListener (p. 1794)

Each concrete **DDSEntity** (p. 1731) is associated with a set of Status objects whose value represents the communication status of that entity. Each status value can be accessed with a corresponding method on the **DDSEntity** (p. 1731).

When these status values change, the corresponding **DDSStatusCondition** (p. 1846) objects are activated and the proper **DDSListener** (p. 1794) objects are invoked to asynchronously inform the application.

An application is notified of communication status by means of the **DDSListener** (p. 1794) or the **DDSWaitSet** (p. 1897) / **DDSCondition** (p. 1544) mechanism. The two mechanisms may be combined in the application (e.g., using **DDSWaitSet** (p. 1897) (s) / **DDSCondition** (p. 1544) (s) to access the data and **DDSListener** (p. 1794) (s) to be warned asynchronously of erroneous communication statuses).

It is likely that the application will choose one or the other mechanism for each particular communication status (not both). However, if both mechanisms are enabled, then the **DDSListener** (p. 1794) mechanism is used first and then the **DDSWaitSet** (p. 1897) objects are signalled.

The statuses may be classified into:

- *read communication statuses*: i.e., those that are related to arrival of data, namely **DDS_DATA_ON_READERS_STATUS** (p. 541) and **DDS_DATA_AVAILABLE_STATUS** (p. 541).

- *plain communication statuses*: i.e., all the others.

Read communication statuses are treated slightly differently than the others because they don't change independently. In other words, at least two changes will appear at the same time (**DDS_DATA_ON_READERS_STATUS** (p. 541) and **DDS_DATA_AVAILABLE_STATUS** (p. 541)) and even several of the last kind may be part of the set. This 'grouping' has to be communicated to the application.

For each plain communication status, there is a corresponding structure to hold the status value. These values contain the information related to the change of status, as well as information related to the statuses themselves (e.g., contains cumulative counts).

"Status Values"

7.135.2 Changes in Status

Associated with each one of an **DDSEntity** (p. 1731)'s communication status is a logical `StatusChangedFlag`. This flag indicates whether that particular communication status has changed since the last time the status was read by the application. The way the status changes is slightly different for the Plain Communication Status and the Read Communication status.

"\p StatusChangedFlag indicates if status has changed"

7.135.2.1 Changes in plain communication status

For the plain communication status, the `StatusChangedFlag` flag is initially set to FALSE. It becomes TRUE whenever the plain communication status changes and it is reset to **DDS_BOOLEAN_FALSE** (p. 513) each time the application accesses the plain communication status via the proper `get_<plain communication status>()` operation on the **DDSEntity** (p. 1731).

The communication status is also reset to FALSE whenever the associated listener operation is called as the listener implicitly accesses the status which is passed as a parameter to the operation. The fact that the status is reset prior to calling the listener means that if the application calls the get_<plain communication status> from inside the listener it will see the status already reset.

An exception to this rule is when the associated listener is the 'nil' listener. The 'nil' listener is treated as a NOOP and the act of calling the 'nil' listener does not reset the communication status.

For example, the value of the `StatusChangedFlag` associated with the **DDS_REQUESTED_DEADLINE_↩ MISSED_STATUS** (p. 540) will become TRUE each time new deadline occurs (which increases **DDS_Requested↩ DeadlineMissedStatus::total_count** (p. 1306)). The value changes to FALSE when the application accesses the status via the corresponding **DDSDataReader::get_requested_deadline_missed_status** (p. 1571) method on the proper Entity

"Changes in \p StatusChangedFlag for plain communication status"

7.135.2.2 Changes in read communication status

For the read communication status, the `StatusChangedFlag` flag is initially set to FALSE. The `Status↩ ChangedFlag` becomes TRUE when either a data-sample arrives or else the **DDS_ViewStateKind** (p. 163), **DDS↩ _SampleStateKind** (p. 161), or **DDS_InstanceStateKind** (p. 165) of any existing sample changes for any reason other than a call to **FooDataReader::read** (p. 1919), **FooDataReader::take** (p. 1920) or their variants. Specifically any of the following events will cause the `StatusChangedFlag` to become TRUE:

- The arrival of new data.
- A change in the **DDS_InstanceStateKind** (p. 165) of a contained instance. This can be caused by either:
  - The arrival of the notification that an instance has been disposed by:
    - the **DDSDataWriter** (p. 1589) that owns it if **OWNERSHIP** (p. 612) QoS kind= **DDS_EXCLUSIVE_↩ OWNERSHIP_QOS** (p. 613)
    - or by any **DDSDataWriter** (p. 1589) if **OWNERSHIP** (p. 612) QoS kind= **DDS_SHARED_↩ OWNERSHIP_QOS** (p. 613)
  - The loss of liveliness of the **DDSDataWriter** (p. 1589) of an instance for which there is no other **DDSData↩ Writer** (p. 1589).
  - The arrival of the notification that an instance has been unregistered by the only **DDSDataWriter** (p. 1589) that is known to be writing the instance.

Depending on the kind of `StatusChangedFlag`, the flag transitions to FALSE again as follows:

- The **DDS_DATA_AVAILABLE_STATUS** (p. 541) `StatusChangedFlag` becomes FALSE when either the corresponding listener operation (on_data_available) is called or the read or take operation (or their variants) is called on the associated **DDSDataReader** (p. 1556).

- The **DDS_DATA_ON_READERS_STATUS** (p. 541) `StatusChangedFlag` becomes FALSE when any of the following events occurs:
  - The corresponding listener operation (on_data_on_readers) is called.
  - The on_data_available listener operation is called on any **DDSDataReader** (p. 1556) belonging to the **DDSSubscriber** (p. 1860).
  - The read or take operation (or their variants) is called on any **DDSDataReader** (p. 1556) belonging to the **DDSSubscriber** (p. 1860).

"Changes in \p StatusChangedFlag for read communication status"

See also

DDSListener (p. 1794)

DDSWaitSet (p. 1897), **DDSCondition** (p. 1544)

7.135.3 Macro Definition Documentation

7.135.3.1 DDS_STATUS_MASK_NONE

```
#define DDS_STATUS_MASK_NONE
```

No bits are set.

Examples

HelloWorld_publisher.cxx, and **HelloWorld_subscriber.cxx**.

7.135.3.2 DDS_STATUS_MASK_ALL

```
#define DDS_STATUS_MASK_ALL
```

All bits are set.

7.135.4 Typedef Documentation

7.135.4.1 DDS_StatusMask

```
typedef  DDS_UnsignedLong  DDS_StatusMask
```

A bit-mask (list) of concrete status types, i.e. **DDS_StatusKind** (p. 538)[].

The bit-mask is an efficient and compact representation of a fixed-length list of **DDS_StatusKind** (p. 538) values.

Bits in the mask correspond to different statuses. You can choose which changes in status will trigger a callback by setting the corresponding status bits in this bit-mask and installing callbacks for each of those statuses.

The bits that are true indicate that the listener will be called back for changes in the corresponding status.

For example:
```
DDS_StatusMask mask = DDS_REQUESTED_DEADLINE_MISSED_STATUS |
                      DDS_DATA_AVAILABLE_STATUS;
datareader->set_listener(listener, mask);
```

or
```
DDS_StatusMask mask = DDS_REQUESTED_DEADLINE_MISSED_STATUS |
                      DDS_DATA_AVAILABLE_STATUS;
datareader = subscriber->create_datareader(topic,
                                DDS_DATAREADER_QOS_DEFAULT,
                                 listener, mask);
```

7.135.5 Enumeration Type Documentation

7.135.5.1 DDS_StatusKind

```
enum  DDS_StatusKind
```

Type for *status* kinds.

Each concrete **DDSEntity** (p. 1731) is associated with a set of `*Status` objects whose values represent the communication status of that **DDSEntity** (p. 1731).

The communication statuses whose changes can be communicated to the application depend on the **DDSEntity** (p. 1731).

Each status value can be accessed with a corresponding method on the **DDSEntity** (p. 1731). The changes on these status values cause activation of the corresponding **DDSStatusCondition** (p. 1846) objects and trigger invocation of the proper **DDSListener** (p. 1794) objects to asynchronously inform the application. Note that not all statuses will activate the **DDSStatusCondition** (p. 1846) or have a corresponding listener callback. Refer to the documentation of the individual statuses for that information.

See also

DDSEntity (p. 1731), **DDSStatusCondition** (p. 1846), **DDSListener** (p. 1794)

Enumerator

DDS_INCONSISTENT_TOPIC_STATUS	Another topic exists with the same name but different characteristics. **Entity:** **DDSTopic** (p. 1885) **Status:** **DDS_InconsistentTopicStatus** (p. 1165) **Listener:** **DDSTopicListener** (p. 1894)
DDS_OFFERED_DEADLINE_MISSED_STATUS	The deadline that the **DDSDataWriter** (p. 1589) has committed through its **DDS_DeadlineQosPolicy** (p. 928) was not respected for a specific instance. **Entity:** **DDSDataWriter** (p. 1589) **QoS:** **DEADLINE** (p. 583) **Status:** **DDS_OfferedDeadlineMissedStatus** (p. 1219) **Listener:** **DDSDataWriterListener** (p. 1612)

Enumerator

DDS_REQUESTED_DEADLINE_MISSED_STATUS	The deadline that the **DDSDataReader** (p. 1556) was expecting through its **DDS_DeadlineQosPolicy** (p. 928) was not respected for a specific instance. **Entity:** **DDSDataReader** (p. 1556) **QoS:** **DEADLINE** (p. 583) **Status:** **DDS_RequestedDeadlineMissedStatus** (p. 1306) **Listener:** **DDSDataReaderListener** (p. 1583)
DDS_OFFERED_INCOMPATIBLE_QOS_STATUS	A QosPolicy value was incompatible with what was requested. **Entity:** **DDSDataWriter** (p. 1589) **Status:** **DDS_OfferedIncompatibleQosStatus** (p. 1221) **Listener:** **DDSDataWriterListener** (p. 1612)
DDS_REQUESTED_INCOMPATIBLE_QOS_STATUS	A QosPolicy value was incompatible with what is offered. **Entity:** **DDSDataReader** (p. 1556) **Status:** **DDS_RequestedIncompatibleQosStatus** (p. 1307) **Listener:** **DDSDataReaderListener** (p. 1583)

Enumerator

DDS_SAMPLE_LOST_STATUS	A sample has been lost (i.e., was never received). **Entity:** **DDSDataReader** (p. 1556) **Status:** **DDS_SampleLostStatus** (p. 1355) **Listener:** **DDSDataReaderListener** (p. 1583)
DDS_SAMPLE_REJECTED_STATUS	A (received) sample has been rejected. **Entity:** **DDSDataReader** (p. 1556) **QoS:** **RESOURCE_LIMITS** (p. 635) **Status:** **DDS_SampleRejectedStatus** (p. 1356) **Listener:** **DDSDataReaderListener** (p. 1583)
DDS_DATA_ON_READERS_STATUS	New data is available. **Entity:** **DDSSubscriber** (p. 1860) **Listener:** **DDSSubscriberListener** (p. 1881)
DDS_DATA_AVAILABLE_STATUS	One or more new data samples have been received. **Entity:** **DDSDataReader** (p. 1556) **Listener:** **DDSDataReaderListener** (p. 1583)

Enumerator

DDS_LIVELINESS_LOST_STATUS	The liveliness that the **DDSDataWriter** (p. 1589) has committed to through its **DDS_LivelinessQosPolicy** (p. 1180) was not respected, thus **DDSDataReader** (p. 1556) entities will consider the **DDSDataWriter** (p. 1589) as no longer alive. **Entity:** **DDSDataWriter** (p. 1589) **QoS:** **LIVELINESS** (p. 607) **Status:** **DDS_LivelinessLostStatus** (p. 1179) **Listener:** **DDSDataWriterListener** (p. 1612)
DDS_LIVELINESS_CHANGED_STATUS	The liveliness of one or more **DDSDataWriter** (p. 1589) that were writing instances read through the **DDSDataReader** (p. 1556) has changed. Some **DDSDataWriter** (p. 1589) have become alive or not_alive. **Entity:** **DDSDataReader** (p. 1556) **QoS:** **LIVELINESS** (p. 607) **Status:** **DDS_LivelinessChangedStatus** (p. 1176) **Listener:** **DDSDataReaderListener** (p. 1583)

Enumerator

DDS_PUBLICATION_MATCHED_STATUS	The **DDSDataWriter** (p. 1589) has found **DDSDataReader** (p. 1556) that matches the **DDSTopic** (p. 1885) and has compatible QoS. **Entity:** **DDSDataWriter** (p. 1589) **Status:** **DDS_PublicationMatchedStatus** (p. 1273) **Listener:** **DDSDataWriterListener** (p. 1612)
DDS_SUBSCRIPTION_MATCHED_STATUS	The **DDSDataReader** (p. 1556) has found **DDSDataWriter** (p. 1589) that matches the **DDSTopic** (p. 1885) and has compatible QoS. **Entity:** **DDSDataReader** (p. 1556) **Status:** **DDS_SubscriptionMatchedStatus** (p. 1379) **Listener:** **DDSDataReaderListener** (p. 1583)
DDS_DATA_WRITER_EMPTY_VIRTUAL_WRITER_↩ STATUS	The status kind for the status when a data writer does not have samples for a virtual writer for which before it had samples.
DDS_INVALID_LOCAL_ADVANCE_NOTICE_STATUS	<<***extension***>> *(p. 432)* The local **DDSDomainParticipant** (p. 1619) has or is about to have an invalid identity or permissions credential. Enables a **DDSDomainParticipant** (p. 1619) callback that is called when the local **DDSDomainParticipant** (p. 1619) has or is about to have an invalid identity or permissions credential. Currently, this status is only triggered when enabling the RTI Security Plugins. Please refer to the `RTI Security Plugins User's Manual` for more information. **Entity:** **DDSDomainParticipant** (p. 1619) **Listener:** **DDSDomainParticipantListener** (p. 1722)

Enumerator

DDS_SERVICE_REQUEST_ACCEPTED_STATUS	<<***extension***>> *(p. 432)* A **DDSDataWriter** (p. 1589) has been issued a **DDS_ServiceRequest** (p. 1359) Enables a **DDSDataWriter** (p. 1589) callback that is called when a **DDS_ServiceRequest** (p. 1359) has been accepted and dispatched to the DataWriter. **Entity:** **DDSDataWriter** (p. 1589) **Listener:** **DDSDataWriterListener** (p. 1612)
DDS_DATA_WRITER_APPLICATION_↩ ACKNOWLEDGMENT_STATUS	<<***extension***>> *(p. 432)* A **DDSDataWriter** (p. 1589) has received an application-level acknowledgment for a sample Enables a **DDSDataWriter** (p. 1589) callback that is called when an application-level acknowledgment from a **DDSDataReader** (p. 1556) is received. The callback is called for each sample that is application-level acknowledged. Changes to this status do not trigger a **DDSStatusCondition** (p. 1846). **Entity:** **DDSDataWriter** (p. 1589) **Listener:** **DDSDataWriterListener** (p. 1612)
DDS_DATA_WRITER_INSTANCE_REPLACED_↩ STATUS	<<***extension***>> *(p. 432)* A **DDSDataWriter** (p. 1589) instance has been replaced Enables a **DDSDataWriter** (p. 1589) callback that is called when an instance in the writer queue is replaced. **Entity:** **DDSDataWriter** (p. 1589) **Listener:** **DDSDataWriterListener** (p. 1612)

Enumerator

DDS_RELIABLE_WRITER_CACHE_CHANGED_↩ STATUS	<<***extension***>> *(p. 432)* The number of unacknowledged samples in a reliable writer's cache has changed such that it has reached a pre-defined trigger point. This status is considered changed at the following times: the cache is empty (i.e., contains no unacknowledged samples), the cache is full (i.e., the sample count has reached the value specified in **DDS_ResourceLimitsQosPolicy::max_samples** (p. 1311)), or the number of samples has reached a high (see **DDS_RtpsReliableWriterProtocol_t::high_↩ watermark** (p. 1322)) or low (see **DDS_RtpsReliableWriterProtocol_t::low_watermark** (p. 1321)) watermark. **Entity:** **DDSDataWriter** (p. 1589) **Status:** **DDS_ReliableWriterCacheChangedStatus** (p. 1302) **Listener:** **DDSDataWriterListener** (p. 1612)
DDS_RELIABLE_READER_ACTIVITY_CHANGED_↩ STATUS	<<***extension***>> *(p. 432)* One or more reliable readers has become active or inactive. A reliable reader is considered active by a reliable writer with which it is matched if that reader acknowledges the samples it has been sent in a timely fashion. For the definition of "timely" in this case, see **DDS_RtpsReliableWriterProtocol_t** (p. 1320) and **DDS_ReliableReaderActivityChangedStatus** (p. 1300). **See also** **DDS_RtpsReliableWriterProtocol_t** (p. 1320) **DDS_ReliableReaderActivityChangedStatus** (p. 1300)
DDS_DATA_WRITER_CACHE_STATUS	<<***extension***>> *(p. 432)* The status of the writer's cache. Changes to this status do not trigger a **DDSStatusCondition** (p. 1846).
DDS_DATA_WRITER_PROTOCOL_STATUS	<<***extension***>> *(p. 432)* The status of a writer's internal protocol related metrics The status of a writer's internal protocol-related metrics, such as the number of samples pushed, pulled, and filtered and the status of wire protocol traffic. Changes to this status information do not trigger a **DDSStatusCondition** (p. 1846).

Enumerator

DDS_DATA_READER_CACHE_STATUS	<<***extension***>> *(p. 432)* The status of the reader's cache. Changes to this status do not trigger a **DDSStatusCondition** (p. 1846).
DDS_DATA_READER_PROTOCOL_STATUS	<<***extension***>> *(p. 432)* The status of a reader's internal protocol related metrics The status of a reader's internal protocol related metrics, like the number of samples received, filtered, rejected; and status of wire protocol traffic. Changes to this status do not trigger a **DDSStatusCondition** (p. 1846).
DDS_DATA_WRITER_SAMPLE_REMOVED_STATUS	<<***extension***>> *(p. 432)* A **DDSDataWriter** (p. 1589) has removed a sample from its queue. Enables a **DDSDataWriter** (p. 1589) callback that is called when a sample is removed from its queue. **Entity:** **DDSDataWriter** (p. 1589) **Listener:** **DDSDataWriterListener** (p. 1612)

7.136 Thread Settings

The properties of a thread of execution. Consult `Platform Notes` for additional platform specific details.

Classes

- struct **DDS_ThreadSettings_t**
  The properties of a thread of execution.

Macros

- #define **DDS_THREAD_SETTINGS_KIND_MASK_DEFAULT**
  The mask of default thread options.

Typedefs

- typedef **DDS_UnsignedLong DDS_ThreadSettingsKindMask**
  *A mask of which each bit is taken from **DDS_ThreadSettingsKind** (p. 548).*

Enumerations

- enum **DDS_ThreadSettingsKind** {
  DDS_THREAD_SETTINGS_FLOATING_POINT ,
  DDS_THREAD_SETTINGS_STDIO ,
  DDS_THREAD_SETTINGS_REALTIME_PRIORITY ,
  DDS_THREAD_SETTINGS_PRIORITY_ENFORCE ,
  DDS_THREAD_SETTINGS_CANCEL_ASYNCHRONOUS }

  A collection of flags used to configure threads of execution.
- enum **DDS_ThreadSettingsCpuRotationKind** {
  DDS_THREAD_SETTINGS_CPU_NO_ROTATION ,
  DDS_THREAD_SETTINGS_CPU_RR_ROTATION }

  *Determines how **DDS_ThreadSettings_t::cpu_list** (p. 1386) affects processor affinity for thread-related QoS policies that apply to multiple threads.*

7.136.1 Detailed Description

The properties of a thread of execution. Consult `Platform Notes` for additional platform specific details.

7.136.2 Macro Definition Documentation

7.136.2.1 DDS_THREAD_SETTINGS_KIND_MASK_DEFAULT

```
#define DDS_THREAD_SETTINGS_KIND_MASK_DEFAULT
```

The mask of default thread options.

7.136.3 Typedef Documentation

7.136.3.1 DDS_ThreadSettingsKindMask

```
typedef DDS_UnsignedLong DDS_ThreadSettingsKindMask
```

A mask of which each bit is taken from **DDS_ThreadSettingsKind** (p. 548).

See also

DDS_ThreadSettings_t (p. 1385)

7.136.4 Enumeration Type Documentation

7.136.4.1 DDS_ThreadSettingsKind

enum **DDS_ThreadSettingsKind**

A collection of flags used to configure threads of execution.

Not all of these options may be relevant for all operating systems. Consult `Platform Notes` for additional details.

See also

DDS_ThreadSettingsKindMask (p. 547)

Enumerator

DDS_THREAD_SETTINGS_FLOATING_POINT	Code executed within the thread may perform floating point operations. Currently applicable only for VxWorks platforms, where user callbacks use floating-point operations.
DDS_THREAD_SETTINGS_STDIO	Code executed within the thread may access standard I/O. Currently applicable only for VxWorks platforms, where user callbacks do standard I/O operations.
DDS_THREAD_SETTINGS_REALTIME_PRIORITY	The thread will be scheduled on a FIFO basis.
DDS_THREAD_SETTINGS_PRIORITY_ENFORCE	Strictly enforce this thread's priority.
DDS_THREAD_SETTINGS_CANCEL_↩ ASYNCHRONOUS	Allows the thread to be cancelled without first reaching a cancellable state or cancellation point.

7.136.4.2 DDS_ThreadSettingsCpuRotationKind

enum **DDS_ThreadSettingsCpuRotationKind**

Determines how **DDS_ThreadSettings_t::cpu_list** (p. 1386) affects processor affinity for thread-related QoS policies that apply to multiple threads.

7.136.5 Controlling CPU Core Affinity for RTI Threads

Most thread-related QoS settings apply to a single thread (such as for the **DDS_EventQosPolicy** (p. 1150), **DDS_↩ DatabaseQosPolicy** (p. 831), and **DDS_AsynchronousPublisherQosPolicy** (p. 798)). However, the thread settings in the **DDS_ReceiverPoolQosPolicy** (p. 1293) control every receive thread created. In this case, there are several schemes to map M threads to N processors; the rotation kind controls which scheme is used.

Controlling CPU Core Affinity is only relevant to the **DDS_ReceiverPoolQosPolicy** (p. 1293). It is ignored within other QoS policies that include **DDS_ThreadSettings_t** (p. 1385).

If **DDS_ThreadSettings_t::cpu_list** (p. 1386) is empty, the rotation is irrelevant since no affinity adjustment will occur. Suppose instead that **DDS_ThreadSettings_t::cpu_list** (p. 1386) = {0, 1} and that the middleware creates three receive threads: {A, B, C}. If **DDS_ThreadSettings_t::cpu_rotation** (p. 1387) is **DDS_THREAD_SETTINGS_CPU_↩ NO_ROTATION** (p. 549), threads A, B and C will have the same processor affinities (0-1), and the OS will control thread scheduling within this bound. It is common to denote CPU affinities as a bitmask, where set bits represent allowed processors to run on. This mask is printed in hex, so a CPU core affinity of 0-1 can be represented by the mask 0x3.

If **DDS_ThreadSettings_t::cpu_rotation** (p. 1387) is **DDS_THREAD_SETTINGS_CPU_RR_ROTATION** (p. 549), each thread will be assigned in round-robin fashion to one of the processors in **DDS_ThreadSettings_t::cpu_list** (p. 1386); perhaps thread A to 0, B to 1, and C to 0. Note that the order in which internal middleware threads spawn is unspecified.

Not all of these options may be relevant for all operating systems. Refer to the Platform Notes for further information.

Enumerator

DDS_THREAD_SETTINGS_CPU_NO_ROTATION	Any thread controlled by this QoS can run on any listed processor, as determined by OS scheduling.
DDS_THREAD_SETTINGS_CPU_RR_ROTATION	Threads controlled by this QoS will be assigned one processor from the list in round-robin order.

7.137 QoS

Quality of Service (QoS) policies.

Modules

- **All QoS Policies**

  All Quality of Service (QoS) policies.
- **QoS by Category**

  Quality of Service (QoS) organized into categories.

7.137.1 Detailed Description

Quality of Service (QoS) policies.

7.138 All QoS Policies

All Quality of Service (QoS) policies.

Modules

- **ASYNCHRONOUS_PUBLISHER**

  *<<**extension**>> (p. 213) Specifies the asynchronous publishing settings of the **DDSPublisher** (p. 1818) instances.*
- **AVAILABILITY**

  *<<**extension**>> (p. 213) Configures the availability of data.*
- **BATCH**

  *<<**extension**>> (p. 213) Batch QoS policy used to enable batching in **DDSDataWriter** (p. 1589) instances.*
- **DATABASE**

  *<<**extension**>> (p. 213) Various threads and resource limits settings used by RTI Connext to control its internal database.*
- **DATA_READER_PROTOCOL**

  *<<**extension**>> (p. 213) Specifies the DataReader-specific protocol QoS.*
- **DATA_READER_RESOURCE_LIMITS**

  *<<**extension**>> (p. 213) Various settings that configure how DataReaders allocate and use physical memory for internal resources.*
- **DATA_REPRESENTATION**

  *<<**spec**>> (p. 209) A list of data representations and compression methods supported by a **DDSDataWriter** (p. 1589) or **DDSDataReader** (p. 1556).*
- **DATA_TAG**

  *<<**spec**>> (p. 209) Stores (name, value) pairs that can be used to determine access permissions.*
- **DATA_WRITER_PROTOCOL**

  *<<**extension**>> (p. 213) Along with **DDS_WireProtocolQosPolicy** (p. 1511) and **DDS_DataReaderProtocolQos**↩**Policy** (p. 843), this QoS policy configures the DDS on-the-network protocol (RTPS).*
- **DATA_WRITER_RESOURCE_LIMITS**

  *<<**extension**>> (p. 213) Various settings that configure how a **DDSDataWriter** (p. 1589) allocates and uses physical memory for internal resources.*
- **DATA_WRITER_TRANSFER_MODE**

  *<<**extension**>> (p. 213) Specifies the DataWriter transfer mode QoS.*
- **DEADLINE**

  *<<**spec**>> (p. 209) Expresses the maximum duration (deadline) within which an instance is expected to be updated.*
- **DESTINATION_ORDER**

  *<<**spec**>> (p. 209) Controls the criteria used to determine the logical order among changes made by **DDSPublisher** (p. 1818) entities to the same instance of data (i.e., matching **DDSTopic** (p. 1885) and key).*
- **DISCOVERY**

  *<<**extension**>> (p. 213) Specifies the attributes required to discover participants in the domain.*
- **DISCOVERY_CONFIG**

  *<<**extension**>> (p. 213) Specifies the discovery configuration QoS.*
- **DOMAIN_PARTICIPANT_RESOURCE_LIMITS**

  *<<**extension**>> (p. 213) Various settings that configure how a **DDSDomainParticipant** (p. 1619) allocates and uses physical memory for internal resources, including the maximum sizes of various properties.*
- **DURABILITY**

  *<<**spec**>> (p. 209) This QoS policy specifies whether or not RTI Connext will store and deliver previously published data samples to new **DDSDataReader** (p. 1556) entities that join the network later.*
- **DURABILITY_SERVICE**

  *<<**spec**>> (p. 209) Various settings to configure the external RTI Persistence Service used by RTI Connext for Data↩Writers with a **DDS_DurabilityQosPolicy** (p. 1005) setting of **DDS_PERSISTENT_DURABILITY_QOS** (p. 599) or **DDS**↩**_TRANSIENT_DURABILITY_QOS** (p. 599).*
- **ENTITY_FACTORY**

<<***spec***>> (p. 209) A QoS policy for all ***DDSEntity*** (p. 1731) types that can act as factories for one or more other ***DDSEntity*** (p. 1731) types.

- **ENTITY_NAME**

  <<***extension***>> (p. 213) Assigns a name to a ***DDSDomainParticipant*** (p. 1619), ***DDSPublisher*** (p. 1818), ***DDSSubscriber*** (p. 1860), ***DDSDataWriter*** (p. 1589) or ***DDSDataReader*** (p. 1556). Except for ***DDSPublisher*** (p. 1818) and ***DDSSubscriber*** (p. 1860), these names are visible during the discovery process and in RTI tools to help you visualize and debug your system.

- **EVENT**

  <<***extension***>> (p. 213) Configures the internal thread in a DomainParticipant that handles timed events.

- **HISTORY**

  <<***spec***>> (p. 209) Specifies the behavior of RTI Connext in the case where the value of an instance changes (one or more times) before it can be successfully communicated to one or more existing subscribers.

- **GROUP_DATA**

  Attaches a buffer of opaque data that is distributed by means of ***Built-in Topics*** (p. 61) during discovery.

- **LATENCY_BUDGET**

  <<***spec***>> (p. 209) Provides a hint as to the maximum acceptable delay from the time the data is written to the time it is received by the subscribing applications.

- **LIFESPAN**

  <<***spec***>> (p. 209) Specifies how long the data written by the ***DDSDataWriter*** (p. 1589) is considered valid.

- **LIVELINESS**

  <<***spec***>> (p. 209) Specifies and configures the mechanism that allows ***DDSDataReader*** (p. 1556) entities to detect when ***DDSDataWriter*** (p. 1589) entities become disconnected or "dead."

- **LOCATORFILTER**

  <<***extension***>> (p. 213) The QoS policy used to report the configuration of a MultiChannel DataWriter as part of ***DDS_PublicationBuiltinTopicData*** (p. 1264).

- **LOGGING**

  <<***extension***>> (p. 213) Configures the RTI Connext logging facility.

- **MONITORING**

  <<***extension***>> (p. 213) Configures the use of RTI Monitoring Library 2.0 to collect and distribute RTI Connext telemetry data.

- **MULTICHANNEL**

  <<***extension***>> (p. 213) Configures the ability of a DataWriter to send data on different multicast groups (addresses) based on the value of the data.

- **OWNERSHIP**

  <<***spec***>> (p. 209) Specifies whether it is allowed for multiple ***DDSDataWriter*** (p. 1589) (s) to write the same instance of the data and if so, how these modifications should be arbitrated.

- **OWNERSHIP_STRENGTH**

  <<***spec***>> (p. 209) Specifies the value of the strength used to arbitrate among multiple ***DDSDataWriter*** (p. 1589) objects that attempt to modify the same instance of a data type (identified by ***DDSTopic*** (p. 1885) + key).

- **PARTITION**

  <<***spec***>> (p. 209) Set of strings that introduces logical partitions in ***DDSDomainParticipant*** (p. 1619), ***DDSPublisher*** (p. 1818), or ***DDSSubscriber*** (p. 1860) entities.

- **PRESENTATION**

  <<***spec***>> (p. 209) Specifies how the samples representing changes to data instances are presented to a subscribing application.

- **PROFILE**

  <<***extension***>> (p. 213) Configures the way that XML documents containing QoS profiles are loaded by RTI Connext.

- **PROPERTY**

  <<***extension***>> (p. 213) Stores (name, value) pairs that can be used to configure certain parameters of RTI Connext that are not exposed through formal QoS policies. Can also be used to store and propagate application-specific name/value pairs that can be retrieved by user code during discovery.

- **PUBLISH_MODE**

  *<<**extension**>> (p. 213) Specifies how RTI Connext sends application data on the network. This QoS policy can be used to tell RTI Connext to use its own thread to send data, instead of the user thread.*
- **READER_DATA_LIFECYCLE**

  *<<**spec**>> (p. 209) Controls how a DataReader manages the lifecycle of the data that it has received.*
- **RECEIVER_POOL**

  *<<**extension**>> (p. 213) Configures threads used by RTI Connext to receive and process data from transports (for example, UDP sockets).*
- **RELIABILITY**

  *<<**spec**>> (p. 209) Indicates the level of reliability offered/requested by RTI Connext.*
- **RESOURCE_LIMITS**

  *<<**spec**>> (p. 209) Controls the amount of physical memory allocated for DDS entities, if dynamic allocations are allowed, and how they occur. Also controls memory usage among different instance values for keyed topics.*
- **SERVICE**

  *<<**extension**>> (p. 213) Service QoS policy is used to indicate what kind of service or tool is associated with the DDS entity.*
- **SYSTEM_RESOURCE_LIMITS**

  *<<**extension**>> (p. 213) Configures DomainParticipant-independent resources used by RTI Connext.*
- **TIME_BASED_FILTER**

  *<<**spec**>> (p. 209) Filter that allows a **DDSDataReader** (p. 1556) to specify that it is interested only in (potentially) a subset of the values of the data.*
- **TOPIC_DATA**

  *Attaches a buffer of opaque data that is distributed by means of **Built-in Topics** (p. 61) during discovery.*
- **TOPIC_QUERY_DISPATCH**

  *<<**extension**>> (p. 213) Configures the ability of a **DDSDataWriter** (p. 1589) to publish historical samples*
- **TRANSPORT_BUILTIN**

  *<<**extension**>> (p. 213) Specifies which built-in transports are used.*
- **TRANSPORT_MULTICAST**

  *<<**extension**>> (p. 213) Specifies the multicast address on which a **DDSDataReader** (p. 1556) wants to receive its data. It can also specify a port number, as well as a subset of the available (at the **DDSDomainParticipant** (p. 1619) level) transports with which to receive the multicast data.*
- **TRANSPORT_MULTICAST_MAPPING**

  *<<**extension**>> (p. 213) Specifies a list of topic expressions and addresses that can be used by an Entity with a specific topic name to receive data.*
- **TRANSPORT_PRIORITY**

  *<<**spec**>> (p. 209) This QoS policy allows the application to take advantage of transport that are capable of sending messages with different priorities.*
- **TRANSPORT_SELECTION**

  *<<**extension**>> (p. 213) Specifies the physical transports that a **DDSDataWriter** (p. 1589) or **DDSDataReader** (p. 1556) may use to send or receive data.*
- **TRANSPORT_UNICAST**

  *<<**extension**>> (p. 213) Specifies a subset of transports and a port number that can be used by an Entity to receive data.*
- **TYPE_CONSISTENCY_ENFORCEMENT**

  *<<**spec**>> (p. 209) Defines the rules for determining whether the type used to publish a given topic is consistent with that used to subscribe to it.*
- **TYPESUPPORT**

  ***[DEPRECATED]** (p. **384**) <<**extension**>> (p. 213) Allows you to attach application-specific values to a **DDSDataWriter** (p. 1589) or **DDSDataReader** (p. 1556), which are passed to the serialization or deserialization routine of the associated data type and choose whether to set padding bytes to zero during serialization.*

- **USER_DATA**

  *Attaches a buffer of opaque data that is distributed by means of **Built-in Topics** (p. 61) during discovery.*
- **WRITER_DATA_LIFECYCLE**

  *<<**spec**>> (p. 209) Controls how a DataWriter handles the lifecycle of the instances (keys) that it is registered to manage.*
- **WIRE_PROTOCOL**

  *<<**extension**>> (p. 213) Specifies the wire protocol related attributes for the **DDSDomainParticipant** (p. 1619).*
- **Extended Qos Support**

  *<<**extension**>> (p. 213) Types and defines used in extended QoS policies.*

Classes

- struct **DDS_QosPrintFormat**

  A collection of attributes used to configure how a QoS appears when printed.
- struct **DDS_QosPolicyCount**

  *Type to hold a counter for a **DDS_QosPolicyId_t** (p. 557).*
- struct **DDS_QosPolicyCountSeq**

  *Declares IDL `sequence` < **DDS_QosPolicyCount** (p. 1283) >*

Macros

- #define **DDS_QosPrintFormat_INITIALIZER**

  *Static initializer for **DDS_QosPrintFormat** (p. 1285).*
- #define **DDS_QOS_POLICY_COUNT**

  *Number of QoS policies in **DDS_QosPolicyId_t** (p. 557).*

Enumerations

- enum **DDS_QosPolicyId_t** {
  DDS_INVALID_QOS_POLICY_ID ,
  DDS_USERDATA_QOS_POLICY_ID ,
  DDS_DURABILITY_QOS_POLICY_ID ,
  DDS_PRESENTATION_QOS_POLICY_ID ,
  DDS_DEADLINE_QOS_POLICY_ID ,
  DDS_LATENCYBUDGET_QOS_POLICY_ID ,
  DDS_OWNERSHIP_QOS_POLICY_ID ,
  DDS_OWNERSHIPSTRENGTH_QOS_POLICY_ID ,
  DDS_LIVELINESS_QOS_POLICY_ID ,
  DDS_TIMEBASEDFILTER_QOS_POLICY_ID ,
  DDS_PARTITION_QOS_POLICY_ID ,
  DDS_RELIABILITY_QOS_POLICY_ID ,
  DDS_DESTINATIONORDER_QOS_POLICY_ID ,
  DDS_HISTORY_QOS_POLICY_ID ,
  DDS_RESOURCELIMITS_QOS_POLICY_ID ,
  DDS_ENTITYFACTORY_QOS_POLICY_ID ,
  DDS_WRITERDATALIFECYCLE_QOS_POLICY_ID ,
  DDS_READERDATALIFECYCLE_QOS_POLICY_ID ,
  DDS_TOPICDATA_QOS_POLICY_ID ,

DDS_GROUPDATA_QOS_POLICY_ID ,
DDS_TRANSPORTPRIORITY_QOS_POLICY_ID ,
DDS_LIFESPAN_QOS_POLICY_ID ,
DDS_DURABILITYSERVICE_QOS_POLICY_ID ,
DDS_DATA_REPRESENTATION_QOS_POLICY_ID ,
DDS_TYPE_CONSISTENCY_ENFORCEMENT_QOS_POLICY_ID ,
DDS_DATATAG_QOS_POLICY_ID ,
DDS_WIREPROTOCOL_QOS_POLICY_ID ,
DDS_DISCOVERY_QOS_POLICY_ID ,
DDS_DATAREADERRESOURCELIMITS_QOS_POLICY_ID ,
DDS_DATAWRITERRESOURCELIMITS_QOS_POLICY_ID ,
DDS_DATAREADERPROTOCOL_QOS_POLICY_ID ,
DDS_DATAWRITERPROTOCOL_QOS_POLICY_ID ,
DDS_DOMAINPARTICIPANTRESOURCELIMITS_QOS_POLICY_ID ,
DDS_EVENT_QOS_POLICY_ID ,
DDS_DATABASE_QOS_POLICY_ID ,
DDS_RECEIVERPOOL_QOS_POLICY_ID ,
DDS_DISCOVERYCONFIG_QOS_POLICY_ID ,
DDS_USEROBJECT_QOS_POLICY_ID = 1013 ,
DDS_SYSTEMRESOURCELIMITS_QOS_POLICY_ID ,
DDS_TRANSPORTSELECTION_QOS_POLICY_ID ,
DDS_TRANSPORTUNICAST_QOS_POLICY_ID ,
DDS_TRANSPORTMULTICAST_QOS_POLICY_ID ,
DDS_TRANSPORTBUILTIN_QOS_POLICY_ID ,
DDS_TYPESUPPORT_QOS_POLICY_ID ,
DDS_PROPERTY_QOS_POLICY_ID ,
DDS_PUBLISHMODE_QOS_POLICY_ID ,
DDS_ASYNCHRONOUSPUBLISHER_QOS_POLICY_ID ,
DDS_ENTITYNAME_QOS_POLICY_ID ,
DDS_SERVICE_QOS_POLICY_ID = 1025 ,
DDS_BATCH_QOS_POLICY_ID ,
DDS_PROFILE_QOS_POLICY_ID ,
DDS_LOCATORFILTER_QOS_POLICY_ID ,
DDS_MULTICHANNEL_QOS_POLICY_ID ,
DDS_TRANSPORTENCAPSULATION_QOS_POLICY_ID = 1030 ,
DDS_PUBLISHERPROTOCOL_QOS_POLICY_ID = 1031 ,
DDS_SUBSCRIBERPROTOCOL_QOS_POLICY_ID = 1032 ,
DDS_TOPICPROTOCOL_QOS_POLICY_ID = 1033 ,
DDS_DOMAINPARTICIPANTPROTOCOL_QOS_POLICY_ID = 1034 ,
DDS_AVAILABILITY_QOS_POLICY_ID ,
DDS_TRANSPORTMULTICASTMAPPING_QOS_POLICY_ID ,
DDS_LOGGING_QOS_POLICY_ID ,
DDS_TOPICQUERYDISPATCH_QOS_POLICY_ID ,
DDS_DATAWRITERTRANSFERMODE_QOS_POLICY_ID ,
DDS_MONITORING_QOS_POLICY_ID }

Type to identify `QosPolicies`.

7.138.1 Detailed Description

All Quality of Service (QoS) policies.

Data Distribution Service (DDS) relies on the use of QoS. A QoS is a set of characteristics that controls some aspect of the behavior of DDS. A QoS is comprised of individual QoS policies (objects conceptually deriving from an *abstract* `QosPolicy` class).

"Supported QoS policies"

The QosPolicy provides the basic mechanism for an application to specify quality of service parameters. It has an attribute name that is used to uniquely identify each `QosPolicy`.

QosPolicy implementation is comprised of a name, an ID, and a type. The type of a `QosPolicy` value may be atomic, such as an integer or float, or compound (a structure). Compound types are used whenever multiple parameters must be set coherently to define a consistent value for a QosPolicy.

QoS (i.e., a list of `QosPolicy` objects) may be associated with all **DDSEntity** (p. 1731) objects in the system such as **DDSTopic** (p. 1885), **DDSDataWriter** (p. 1589), **DDSDataReader** (p. 1556), **DDSPublisher** (p. 1818), **DDSSubscriber** (p. 1860), and **DDSDomainParticipant** (p. 1619).

7.138.2 Specifying QoS on entities

QosPolicies can be set programmatically when an **DDSEntity** (p. 1731) is created, or modified with the **DDSEntity** (p. 1731)'s **set_qos (abstract)** (p. 1732) method.

QosPolicies can also be configured from XML resources (files, strings). With this approach, you can change the QoS without recompiling the application. For more information, see **Configuring QoS Profiles with XML** (p. 390).

To customize a **DDSEntity** (p. 1731)'s QoS before creating the entity, the correct pattern is:

- First, initialize a QoS object with the appropriate INITIALIZER constructor.
- Call the relevant get_<entity>_default_qos() method.
- Modify the QoS values as desired.
- Finally, create the entity.

Each `QosPolicy` is treated independently from the others. This approach has the advantage of being very extensible. However, there may be cases where several policies are in conflict. Consistency checking is performed each time the policies are modified via the **set_qos (abstract)** (p. 1732) operation, or when the **DDSEntity** (p. 1731) is created.

When a policy is changed after being set to a given value, it is not required that the new value be applied instantaneously; RTI Connext is allowed to apply it after a transition phase. In addition, some `QosPolicy` have immutable semantics, meaning that they can only be specified either at **DDSEntity** (p. 1731) creation time or else prior to calling the **DDSEntity::enable** (p. 1734) operation on the entity.

Each **DDSEntity** (p. 1731) can be configured with a list of `QosPolicy` objects. However, not all QosPolicies are supported by each **DDSEntity** (p. 1731). For instance, a **DDSDomainParticipant** (p. 1619) supports a different set of QosPolicies than a **DDSTopic** (p. 1885) or a **DDSPublisher** (p. 1818).

Additional properties that are not exposed through the formal QoS policies can also be set for an **DDSEntity** (p. 1731) via the **DDS_PropertyQosPolicy** (p. 1260). These properties are described in the `Property Reference Guide`.

7.138.3 QoS compatibility

In several cases, for communications to occur properly (or efficiently), a QosPolicy on the publisher side must be compatible with a corresponding policy on the subscriber side. For example, if a **DDSSubscriber** (p. 1860) requests to receive data reliably while the corresponding **DDSPublisher** (p. 1818) defines a best-effort policy, communication will not happen as requested.

To address this issue and maintain the desirable decoupling of publication and subscription as much as possible, the QosPolicy specification follows the **subscriber-requested, publisher-offered pattern**.

In this pattern, the subscriber side can specify a "requested" value for a particular `QosPolicy`. The publisher side specifes an "offered" value for that `QosPolicy`. RTI Connext will then determine whether the value requested by the subscriber side is compatible with what is offered by the publisher side. If the two policies are compatible, then communication will be established. If the two policies are not compatible, RTI Connext will not establish communications between the two **DDSEntity** (p. 1731) objects and will record this fact by means of the **DDS_OFFERED_INCOMPATIBLE_↩ QOS_STATUS** (p. 540) on the publisher end and **DDS_REQUESTED_INCOMPATIBLE_QOS_STATUS** (p. 540) on the subscriber end. The application can detect this fact by means of a **DDSListener** (p. 1794) or a **DDSCondition** (p. 1544).

The following **properties** are defined on a QosPolicy.

- **RxO (**p. **556) property**
  The QosPolicy objects that need to be set in a compatible manner between the **publisher** and **subscriber** end are indicated by the setting of the **RxO** (p. 556) property:
  - **RxO** (p. 556) = **YES**
    indicates that the policy can be set both at the publishing and subscribing ends and the values must be set in a compatible manner. In this case the compatible values are explicitly defined.
  - **RxO** (p. 556) = **NO**
    indicates that the policy can be set both at the publishing and subscribing ends but the two settings are independent. That is, all combinations of values are compatible.
  - **RxO** (p. 556) = **N/A**
    indicates that the policy can only be specified at either the publishing or the subscribing end, but not at both ends. So compatibility does not apply.
- **Changeable (**p. **556)property**
  Determines whether a QosPolicy can be changed.
  - **NO (**p. **556)** -- policy can only be specified at **DDSEntity** (p. 1731) creation time.
  - **UNTIL ENABLE (**p. **556)** -- policy can only be changed before the **DDSEntity** (p. 1731) is enabled.
  - **YES (**p. **556)** -- policy can be changed at any time.

7.138.4 Macro Definition Documentation

7.138.4.1 DDS_QosPrintFormat_INITIALIZER

```
#define DDS_QosPrintFormat_INITIALIZER
```

Static initializer for **DDS_QosPrintFormat** (p. 1285).

Use this initializer to ensure that new objects do not have uninitialized contents.
```
struct DDS_QosPrintFormat format = DDS_QosPrintFormat_INITIALIZER;
```

See also

DDS_QosPrintFormat (p. 1285)

7.138.4.2 DDS_QOS_POLICY_COUNT

```
#define DDS_QOS_POLICY_COUNT
```

Number of QoS policies in **DDS_QosPolicyId_t** (p. 557).

7.138.5 Enumeration Type Documentation

7.138.5.1 DDS_QosPolicyId_t

```
enum DDS_QosPolicyId_t
```

Type to identify `QosPolicies`.

Enumerator

DDS_INVALID_QOS_POLICY_ID	Identifier for an invalid QoS policy.
DDS_USERDATA_QOS_POLICY_ID	<<***spec***>> *(p. 209)* Identifier for **DDS_UserDataQosPolicy** (p. 1504)
DDS_DURABILITY_QOS_POLICY_ID	<<***spec***>> *(p. 209)* Identifier for **DDS_DurabilityQosPolicy** (p. 1005)
DDS_PRESENTATION_QOS_POLICY_ID	<<***spec***>> *(p. 209)* Identifier for **DDS_PresentationQosPolicy** (p. 1249)
DDS_DEADLINE_QOS_POLICY_ID	<<***spec***>> *(p. 209)* Identifier for **DDS_DeadlineQosPolicy** (p. 928)
DDS_LATENCYBUDGET_QOS_POLICY_ID	<<***spec***>> *(p. 209)* Identifier for **DDS_LatencyBudgetQosPolicy** (p. 1173)
DDS_OWNERSHIP_QOS_POLICY_ID	<<***spec***>> *(p. 209)* Identifier for **DDS_OwnershipQosPolicy** (p. 1222)

Enumerator

DDS_OWNERSHIPSTRENGTH_QOS_POLICY_ID	<<***spec***>> *(p. 209)* Identifier for **DDS_OwnershipStrengthQosPolicy** (p. 1227)
DDS_LIVELINESS_QOS_POLICY_ID	<<***spec***>> *(p. 209)* Identifier for **DDS_LivelinessQosPolicy** (p. 1180)
DDS_TIMEBASEDFILTER_QOS_POLICY_ID	<<***spec***>> *(p. 209)* Identifier for **DDS_TimeBasedFilterQosPolicy** (p. 1389)
DDS_PARTITION_QOS_POLICY_ID	<<***spec***>> *(p. 209)* Identifier for **DDS_PartitionQosPolicy** (p. 1240)
DDS_RELIABILITY_QOS_POLICY_ID	<<***spec***>> *(p. 209)* Identifier for **DDS_ReliabilityQosPolicy** (p. 1295)
DDS_DESTINATIONORDER_QOS_POLICY_ID	<<***spec***>> *(p. 209)* Identifier for **DDS_DestinationOrderQosPolicy** (p. 931)
DDS_HISTORY_QOS_POLICY_ID	<<***spec***>> *(p. 209)* Identifier for **DDS_HistoryQosPolicy** (p. 1162)
DDS_RESOURCELIMITS_QOS_POLICY_ID	<<***spec***>> *(p. 209)* Identifier for **DDS_ResourceLimitsQosPolicy** (p. 1308)
DDS_ENTITYFACTORY_QOS_POLICY_ID	<<***spec***>> *(p. 209)* Identifier for **DDS_EntityFactoryQosPolicy** (p. 1145)
DDS_WRITERDATALIFECYCLE_QOS_POLICY_ID	<<***spec***>> *(p. 209)* Identifier for **DDS_WriterDataLifecycleQosPolicy** (p. 1524)
DDS_READERDATALIFECYCLE_QOS_POLICY_ID	<<***spec***>> *(p. 209)* Identifier for **DDS_ReaderDataLifecycleQosPolicy** (p. 1289)
DDS_TOPICDATA_QOS_POLICY_ID	<<***spec***>> *(p. 209)* Identifier for **DDS_TopicDataQosPolicy** (p. 1396)
DDS_GROUPDATA_QOS_POLICY_ID	<<***spec***>> *(p. 209)* Identifier for **DDS_GroupDataQosPolicy** (p. 1160)
DDS_TRANSPORTPRIORITY_QOS_POLICY_ID	<<***spec***>> *(p. 209)* Identifier for **DDS_TransportPriorityQosPolicy** (p. 1419)
DDS_LIFESPAN_QOS_POLICY_ID	<<***spec***>> *(p. 209)* Identifier for **DDS_LifespanQosPolicy** (p. 1175)
DDS_DURABILITYSERVICE_QOS_POLICY_ID	<<***spec***>> *(p. 209)* Identifier for **DDS_DurabilityServiceQosPolicy** (p. 1009)
DDS_DATA_REPRESENTATION_QOS_POLICY_ID	<<***spec***>> *(p. 209)* Identifier for **DDS_DataRepresentationQosPolicy** (p. 886)
DDS_TYPE_CONSISTENCY_ENFORCEMENT_QOS_POLICY_ID	<<***spec***>> *(p. 209)* Identifier for **DDS_TypeConsistencyEnforcementQosPolicy** (p. 1493)
DDS_DATATAG_QOS_POLICY_ID	Identifier for **DDS_DataTagQosPolicy** (p. 575).
DDS_WIREPROTOCOL_QOS_POLICY_ID	<<***extension***>> *(p. 213)* Identifier for **DDS_WireProtocolQosPolicy** (p. 1511)
DDS_DISCOVERY_QOS_POLICY_ID	<<***extension***>> *(p. 213)* Identifier for **DDS_DiscoveryQosPolicy** (p. 960)
DDS_DATAREADERRESOURCELIMITS_QOS_POLICY_ID	<<***extension***>> *(p. 213)* Identifier for **DDS_DataReaderResourceLimitsQosPolicy** (p. 867)
DDS_DATAWRITERRESOURCELIMITS_QOS_POLICY_ID	<<***extension***>> *(p. 213)* Identifier for **DDS_DataWriterResourceLimitsQosPolicy** (p. 917)

Enumerator

DDS_DATAREADERPROTOCOL_QOS_POLICY_ID	<<***extension***>> *(p. 213)* Identifier for **DDS_DataReaderProtocolQosPolicy** (p. 843)
DDS_DATAWRITERPROTOCOL_QOS_POLICY_ID	<<***extension***>> *(p. 213)* Identifier for **DDS_DataWriterProtocolQosPolicy** (p. 892)
DDS_DOMAINPARTICIPANTRESOURCELIMITS_↩ QOS_POLICY_ID	<<***extension***>> *(p. 213)* Identifier for **DDS_DomainParticipantResourceLimitsQosPolicy** (p. 977)
DDS_EVENT_QOS_POLICY_ID	<<***extension***>> *(p. 213)* Identifier for **DDS_EventQosPolicy** (p. 1150)
DDS_DATABASE_QOS_POLICY_ID	<<***extension***>> *(p. 213)* Identifier for **DDS_DatabaseQosPolicy** (p. 831)
DDS_RECEIVERPOOL_QOS_POLICY_ID	<<***extension***>> *(p. 213)* Identifier for **DDS_ReceiverPoolQosPolicy** (p. 1293)
DDS_DISCOVERYCONFIG_QOS_POLICY_ID	<<***extension***>> *(p. 213)* Identifier for **DDS_DiscoveryConfigQosPolicy** (p. 934)
DDS_SYSTEMRESOURCELIMITS_QOS_POLICY_ID	<<***extension***>> *(p. 213)* Identifier for **DDS_SystemResourceLimitsQosPolicy** (p. 1382)
DDS_TRANSPORTSELECTION_QOS_POLICY_ID	<<***extension***>> *(p. 213)* Identifier for **DDS_TransportSelectionQosPolicy** (p. 1421)
DDS_TRANSPORTUNICAST_QOS_POLICY_ID	<<***extension***>> *(p. 213)* Identifier for **DDS_TransportUnicastQosPolicy** (p. 1422)
DDS_TRANSPORTMULTICAST_QOS_POLICY_ID	<<***extension***>> *(p. 213)* Identifier for **DDS_TransportMulticastQosPolicy** (p. 1415)
DDS_TRANSPORTBUILTIN_QOS_POLICY_ID	<<***extension***>> *(p. 213)* Identifier for **DDS_TransportBuiltinQosPolicy** (p. 1408)
DDS_TYPESUPPORT_QOS_POLICY_ID	<<***extension***>> *(p. 213)* Identifier for **DDS_TypeSupportQosPolicy** (p. 1499)
DDS_PROPERTY_QOS_POLICY_ID	<<***extension***>> *(p. 213)* Identifier for **DDS_PropertyQosPolicy** (p. 1260)
DDS_PUBLISHMODE_QOS_POLICY_ID	<<***extension***>> *(p. 213)* Identifier for **DDS_PublishModeQosPolicy** (p. 1279)
DDS_ASYNCHRONOUSPUBLISHER_QOS_↩ POLICY_ID	<<***extension***>> *(p. 213)* Identifier for **DDS_AsynchronousPublisherQosPolicy** (p. 798)
DDS_ENTITYNAME_QOS_POLICY_ID	<<***extension***>> *(p. 213)* Identifier for **DDS_EntityNameQosPolicy** (p. 1147)
DDS_BATCH_QOS_POLICY_ID	<<***extension***>> *(p. 213)* Identifier for **DDS_BatchQosPolicy** (p. 810)
DDS_PROFILE_QOS_POLICY_ID	<<***extension***>> *(p. 213)* Identifier for **DDS_ProfileQosPolicy** (p. 1257)
DDS_LOCATORFILTER_QOS_POLICY_ID	<<***extension***>> *(p. 213)* Identifier for **DDS_LocatorFilterQosPolicy** (p. 1186)
DDS_MULTICHANNEL_QOS_POLICY_ID	<<***extension***>> *(p. 213)* Identifier for **DDS_MultiChannelQosPolicy** (p. 1214)
DDS_AVAILABILITY_QOS_POLICY_ID	<<***extension***>> *(p. 213)* Identifier for **DDS_AvailabilityQosPolicy** (p. 805)
DDS_TRANSPORTMULTICASTMAPPING_QOS_↩ POLICY_ID	<<***extension***>> *(p. 213)* Identifier for **DDS_TransportMulticastMappingQosPolicy** (p. 1413)

Enumerator

DDS_LOGGING_QOS_POLICY_ID	<<***extension***>> *(p. 213)* Identifier for **DDS_LoggingQosPolicy** (p. 1188)
DDS_TOPICQUERYDISPATCH_QOS_POLICY_ID	<<***extension***>> *(p. 213)* Identifier for **DDS_TopicQueryDispatchQosPolicy** (p. 1404)
DDS_DATAWRITERTRANSFERMODE_QOS_↩ POLICY_ID	<<***extension***>> *(p. 213)* Identifier for **DDS_DataWriterTransferModeQosPolicy** (p. 927)
DDS_MONITORING_QOS_POLICY_ID	<<***extension***>> *(p. 213)* Identifier for **DDS_MonitoringQosPolicy** (p. 1210)

7.139 QoS by Category

Quality of Service (QoS) organized into categories.

Modules

- **Basic QoS**

  QoS that are considered basic and fundamental to RTI Connext.
- **DDS Specification QoS**

  QoS that are defined in one of the OMG DDS Specifications.
- **DDS Extension QoS**

  QoS that are RTI extensions to the DDS standard.
- **Functional Categories**

  *These categories help you to discover which QoS may be relevant or related to a desired functionality or feature. For example, if you are interested in **Sending Large Data** (p. 290), then looking at any QoS in that category will be helpful.*
- **QoS Mutability**

  Some QoS are mutable after an entity is enabled, and others are not.
- **Other QoS Attributes**

  *These attributes are characteristics of the QoS – for example if they are **mutable** (p. 369) or **immutable** (p. 333), or if they **apply per-instance** (p. 375).*
- **Deprecated QoS**

  Deprecated QoS.

7.139.1 Detailed Description

Quality of Service (QoS) organized into categories.

The QoS have been grouped into a number of different categories.

These categories can help you to discover which QoS may be relevant or related to a desired functionality or feature. For example, if you are interested in **Sending Large Data** (p. 290), then looking at any QoS in that category will be helpful. The different functional categories are listed on **Functional Categories** (p. 252).

Some categories are not necessarily related to achieving a specific functionality, but are characteristics of the QoS – for example if they are **mutable** (p. 369) or **immutable** (p. 333), or if they **apply per-instance** (p. 375). These categories are listed under **QoS Mutability** (p. 333) and **Other QoS Attributes** (p. 375).

7.140 ASYNCHRONOUS_PUBLISHER

<<***extension***>> *(p. 213)* Specifies the asynchronous publishing settings of the **DDSPublisher** (p. 1818) instances.

Classes

- struct **DDS_AsynchronousPublisherQosPolicy**
  <<***extension***>> *(p. 213) Configures the mechanism that sends user data in an external middleware thread.*

Variables

- const char ∗const **DDS_ASYNCHRONOUSPUBLISHER_QOS_POLICY_NAME**
  *Stringified human-readable name for **DDS_AsynchronousPublisherQosPolicy** (p. 798).*

7.140.1 Detailed Description

<<***extension***>> *(p. 213)* Specifies the asynchronous publishing settings of the **DDSPublisher** (p. 1818) instances.

7.140.2 Variable Documentation

7.140.2.1 DDS_ASYNCHRONOUSPUBLISHER_QOS_POLICY_NAME

```
const char* const DDS_ASYNCHRONOUSPUBLISHER_QOS_POLICY_NAME  [extern]
```

Stringified human-readable name for **DDS_AsynchronousPublisherQosPolicy** (p. 798).

7.141 AVAILABILITY

<<***extension***>> *(p. 213)* Configures the availability of data.

Classes

- struct **DDS_EndpointGroup_t**
  Specifies a group of endpoints that can be collectively identified by a name and satisfied by a quorum.
- struct **DDS_EndpointGroupSeq**
  *A sequence of **DDS_EndpointGroup_t** (p. 1142).*
- struct **DDS_AvailabilityQosPolicy**
  <<***extension***>> *(p. 213) Configures the availability of data.*

Variables

- const char ∗const **DDS_AVAILABILITY_QOS_POLICY_NAME**

  *Stringified human-readable name for **DDS_AvailabilityQosPolicy** (p. 805).*

7.141.1 Detailed Description

<<***extension***>> *(p. 213)* Configures the availability of data.

7.141.2 Variable Documentation

7.141.2.1 DDS_AVAILABILITY_QOS_POLICY_NAME

```
const char* const DDS_AVAILABILITY_QOS_POLICY_NAME  [extern]
```

Stringified human-readable name for **DDS_AvailabilityQosPolicy** (p. 805).

7.142 BATCH

<<***extension***>> *(p. 213)* Batch QoS policy used to enable batching in **DDSDataWriter** (p. 1589) instances.

Classes

- struct **DDS_BatchQosPolicy**

  <<***extension***>> *(p. 213) Used to configure batching of multiple samples into a single network packet in order to increase throughput for small samples.*

Variables

- const char ∗const **DDS_BATCH_QOS_POLICY_NAME**

  *Stringified human-readable name for **DDS_BatchQosPolicy** (p. 810).*

7.142.1 Detailed Description

<<***extension***>> *(p. 213)* Batch QoS policy used to enable batching in **DDSDataWriter** (p. 1589) instances.

7.142.2 Variable Documentation

7.142.2.1 DDS_BATCH_QOS_POLICY_NAME

```
const char* const DDS_BATCH_QOS_POLICY_NAME  [extern]
```

Stringified human-readable name for **DDS_BatchQosPolicy** (p. 810).

7.143 DATABASE

<<***extension***>> *(p. 213)* Various threads and resource limits settings used by RTI Connext to control its internal database.

Classes

- struct **DDS_DatabaseQosPolicy**

  <<***extension***>> *(p. 213) Various threads and resource limits settings used by RTI Connext to control its internal database.*

Variables

- const char ∗const **DDS_DATABASE_QOS_POLICY_NAME**

  *Stringified human-readable name for **DDS_DatabaseQosPolicy** (p. 831).*

7.143.1 Detailed Description

<<***extension***>> *(p. 213)* Various threads and resource limits settings used by RTI Connext to control its internal database.

7.143.2 Variable Documentation

7.143.2.1 DDS_DATABASE_QOS_POLICY_NAME

```
const char* const DDS_DATABASE_QOS_POLICY_NAME  [extern]
```

Stringified human-readable name for **DDS_DatabaseQosPolicy** (p. 831).

7.144 DATA_READER_PROTOCOL

<<***extension***>> *(*p. *213)* Specifies the DataReader-specific protocol QoS.

Classes

- struct **DDS_DataReaderProtocolQosPolicy**

  <<***extension***>> *(*p. *213) Along with* ***DDS_WireProtocolQosPolicy*** *(*p. *1511) and* ***DDS_DataWriterProtocolQosPolicy*** *(*p. *892), this QoS policy configures the DDS on-the-network protocol (RTPS).*

Variables

- const char ∗const **DDS_DATAREADERPROTOCOL_QOS_POLICY_NAME**

  Stringified human-readable name for ***DDS_DataReaderProtocolQosPolicy*** *(*p. *843).*

7.144.1 Detailed Description

<<***extension***>> *(*p. *213)* Specifies the DataReader-specific protocol QoS.

7.144.2 Variable Documentation

7.144.2.1 DDS_DATAREADERPROTOCOL_QOS_POLICY_NAME

```
const char* const DDS_DATAREADERPROTOCOL_QOS_POLICY_NAME  [extern]
```

Stringified human-readable name for **DDS_DataReaderProtocolQosPolicy** (p. 843).

7.145 DATA_READER_RESOURCE_LIMITS

<<***extension***>> *(*p. *213)* Various settings that configure how DataReaders allocate and use physical memory for internal resources.

Classes

- struct **DDS_DataReaderResourceLimitsInstanceReplacementSettings**

  Instance replacement kind applied to each instance state.

- struct **DDS_DataReaderResourceLimitsQosPolicy**

  <<***extension***>> *(*p. *213) Various settings that configure how a* ***DDSDataReader*** *(*p. *1556) allocates and uses physical memory for internal resources.*

Enumerations

- enum **DDS_DataReaderInstanceRemovalKind** {
  DDS_NO_INSTANCE_REMOVAL = PRES_NO_INSTANCE_REMOVAL ,
  DDS_EMPTY_INSTANCE_REMOVAL = PRES_EMPTY_INSTANCE_REMOVAL ,
  DDS_FULLY_PROCESSED_INSTANCE_REMOVAL = PRES_FULLY_PROCESSED_INSTANCE_REMOVAL ,
  DDS_ANY_INSTANCE_REMOVAL = PRES_ANY_INSTANCE_REMOVAL }

  *Sets the kinds of instances that can be replaced when instance resource limits (**DDS_ResourceLimitsQosPolicy::max_instances** (p. 1311)) are reached.*

Variables

- const char ∗const **DDS_DATAREADERRESOURCELIMITS_QOS_POLICY_NAME**

  *Stringified human-readable name for **DDS_DataReaderResourceLimitsQosPolicy** (p. 867).*
- const **DDS_Long** **DDS_AUTO_MAX_TOTAL_INSTANCES**

  *<<**extension**>> (p. 213) This value is used to make **DDS_DataReaderResourceLimitsQosPolicy::max_total_instances** (p. 877) equal to **DDS_ResourceLimitsQosPolicy::max_instances** (p. 1311).*

7.145.1 Detailed Description

*<<**extension**>> (p. 213)* Various settings that configure how DataReaders allocate and use physical memory for internal resources.

7.145.2 Enumeration Type Documentation

7.145.2.1 DDS_DataReaderInstanceRemovalKind

`enum` **DDS_DataReaderInstanceRemovalKind**

Sets the kinds of instances that can be replaced when instance resource limits (**DDS_ResourceLimitsQosPolicy::max_instances** (p. 1311)) are reached.

See also

DDS_DataReaderResourceLimitsQosPolicy::instance_replacement (p. 883)

Enumerator

DDS_NO_INSTANCE_REMOVAL	No instance can be removed. If an instance resource is required because **DDS_ResourceLimitsQosPolicy::max_instances** (p. 1311) is reached, this setting will disallow instances from being replaced. Samples for new instances will be dropped and reported as lost with reason **DDS_LOST_BY_INSTANCES_LIMIT** (p. 140).
DDS_EMPTY_INSTANCE_REMOVAL	Only empty instances can be removed. Instances can be replaced only if they are empty. An instance is considered empty when all samples have been taken or removed from the DataReader queue due to the **DDS_LifespanQosPolicy** (p. 1175) or sample purging due to the **DDS_ReaderDataLifecycleQosPolicy** (p. 1289), and there are no outstanding loans on any of the instance's samples.
DDS_FULLY_PROCESSED_INSTANCE_REMOVAL	Only fully-processed instances can be removed. An instance is considered fully processed if every sample for the instance has been processed by the application. A sample is considered processed by the application depending on the **DDS_ReliabilityQosPolicy::kind** (p. 1298): • **DDS_RELIABLE_RELIABILITY_QOS** (p. 633) (depends on the **DDS_ReliabilityQosPolicy↩ AcknowledgmentModeKind** (p. 633)): – **DDS_PROTOCOL_↩ ACKNOWLEDGMENT_MODE** (p. 633) or **DDS_APPLICATION_AUTO_↩ ACKNOWLEDGMENT_MODE** (p. 633): The sample is considered processed when it has been read or taken by the application and return_loan has been called. – **DDS_APPLICATION_EXPLICIT_↩ ACKNOWLEDGMENT_MODE** (p. 634): The sample is considered processed when the subscribing application has explicitly acknowledged the DDS sample, the AppAckConf has been received, and the application has called return_loan. • **DDS_BEST_EFFORT_RELIABILITY_QOS** (p. 633): All samples are considered processed when they have been read or taken by the application and return_loan has been called.
DDS_ANY_INSTANCE_REMOVAL	Any instance can be removed. Instances can be replaced regardless of whether the subscribing application has processed all of the samples. Samples that have not been processed will be removed.

7.145.3 Variable Documentation

7.145.3.1 DDS_DATAREADERRESOURCELIMITS_QOS_POLICY_NAME

```
const char* const DDS_DATAREADERRESOURCELIMITS_QOS_POLICY_NAME  [extern]
```

Stringified human-readable name for **DDS_DataReaderResourceLimitsQosPolicy** (p. 867).

7.145.3.2 DDS_AUTO_MAX_TOTAL_INSTANCES

```
const  DDS_Long DDS_AUTO_MAX_TOTAL_INSTANCES  [extern]
```

<<***extension***>> *(p. 213)* This value is used to make **DDS_DataReaderResourceLimitsQosPolicy::max_total_instances** (p. 877) equal to **DDS_ResourceLimitsQosPolicy::max_instances** (p. 1311).

7.146 DATA_REPRESENTATION

<<***spec***>> *(p. 209)* A list of data representations and compression methods supported by a **DDSDataWriter** (p. 1589) or **DDSDataReader** (p. 1556).

Modules

- **Compression Settings**

  <<***extension***>> *(p. 213) Compression settings that can be applied to data that is sent and received by DataWriters and DataReaders.*

Classes

- struct **DDS_DataRepresentationIdSeq**

  Declares IDL `sequence` *<* ***DDS_DataRepresentationId_t*** *(p. 568) >*
- struct **DDS_DataRepresentationQosPolicy**

  <<***spec***>> *(p. 209) This QoS policy contains a list of representation identifiers and compression settings used by* ***DDSDataWriter*** *(p. 1589) and* ***DDSDataReader*** *(p. 1556) entities to negotiate which data representation and compression settings to use.*

Typedefs

- typedef **DDS_Short DDS_DataRepresentationId_t**

  A two-byte signed integer that identifies a data representation.

Variables

- const **DDS_DataRepresentationId_t DDS_XCDR_DATA_REPRESENTATION**

  Corresponds to the Extended Common Data Representation encoding version 1.
- const **DDS_DataRepresentationId_t DDS_XML_DATA_REPRESENTATION**

  Corresponds to the XML Data Representation (unsupported).
- const **DDS_DataRepresentationId_t DDS_XCDR2_DATA_REPRESENTATION**

  Corresponds to the Extended Common Data Representation encoding version 2.
- const **DDS_DataRepresentationId_t DDS_AUTO_DATA_REPRESENTATION**

  Representation is automatically chosen based on the type.
- const char *const **DDS_DATA_REPRESENTATION_QOS_POLICY_NAME**

  *Stringified human-readable name for **DDS_DataRepresentationQosPolicy** (p. 886).*

7.146.1 Detailed Description

<<***spec***>> *(p. 209)* A list of data representations and compression methods supported by a **DDSDataWriter** (p. 1589) or **DDSDataReader** (p. 1556).

7.146.2 Typedef Documentation

7.146.2.1 DDS_DataRepresentationId_t

```
typedef DDS_Short DDS_DataRepresentationId_t
```

A two-byte signed integer that identifies a data representation.

7.146.3 Variable Documentation

7.146.3.1 DDS_XCDR_DATA_REPRESENTATION

```
const DDS_DataRepresentationId_t DDS_XCDR_DATA_REPRESENTATION [extern]
```

Corresponds to the Extended Common Data Representation encoding version 1.

7.146.3.2 DDS_XML_DATA_REPRESENTATION

```
const DDS_DataRepresentationId_t DDS_XML_DATA_REPRESENTATION [extern]
```

Corresponds to the XML Data Representation (unsupported).

Note

This value is currently not supported.

7.146.3.3 DDS_XCDR2_DATA_REPRESENTATION

```
const DDS_DataRepresentationId_t DDS_XCDR2_DATA_REPRESENTATION [extern]
```

Corresponds to the Extended Common Data Representation encoding version 2.

7.146.3.4 DDS_AUTO_DATA_REPRESENTATION

```
const DDS_DataRepresentationId_t DDS_AUTO_DATA_REPRESENTATION [extern]
```

Representation is automatically chosen based on the type.

For plain language binding, if the allowed_data_representation annotation is not specified or if it contains the value XCDR, RTI Connext translates this field to **DDS_XCDR_DATA_REPRESENTATION** (p. 568). Otherwise, it translates this field to **DDS_XCDR2_DATA_REPRESENTATION** (p. 569).

For the **FlatData language binding** (p. 776), RTI Connext translates this field to **DDS_XCDR2_DATA_↩ REPRESENTATION** (p. 569).

Examples

HelloWorldPlugin.cxx.

7.146.3.5 DDS_DATA_REPRESENTATION_QOS_POLICY_NAME

```
const char* const DDS_DATA_REPRESENTATION_QOS_POLICY_NAME [extern]
```

Stringified human-readable name for **DDS_DataRepresentationQosPolicy** (p. 886).

7.147 Compression Settings

<<***extension***>> *(p. 213)* Compression settings that can be applied to data that is sent and received by DataWriters and DataReaders.

Classes

- struct **DDS_CompressionSettings_t**

  <<***extension***>> *(p. 213) Settings related to compressing user data.*

Macros

- #define **DDS_COMPRESSION_ID_MASK_NONE** ((**DDS_CompressionIdMask**) (0))

  DDS_CompressionIdMask *(p. 573) used to disable user-data compression for an endpoint.*
- #define **DDS_COMPRESSION_ID_MASK_ALL**

  DDS_CompressionIdMask *(p. 573) mask with all the suported built-in compression algorithms enabled.*
- #define **DDS_COMPRESSION_LEVEL_BEST_COMPRESSION** (10)

  The best compression ratio possible for a compression algorithm.
- #define **DDS_COMPRESSION_LEVEL_BEST_SPEED** (1)

  The best compression speed possible for a compression algorithm.
- #define **DDS_COMPRESSION_LEVEL_DEFAULT DDS_COMPRESSION_LEVEL_BEST_COMPRESSION**

  Default level of compression.
- #define **DDS_COMPRESSION_THRESHOLD_DEFAULT** 8192

  Default threshold from which a serialized sample will be eligible to be compressed.
- #define **DDS_COMPRESSION_ID_MASK_PUBLICATION_DEFAULT DDS_COMPRESSION_ID_MASK_↩ NONE**

  DDS_CompressionIdMask *(p. 573) mask with the default value for publication.*
- #define **DDS_COMPRESSION_ID_MASK_SUBSCRIPTION_DEFAULT**

  DDS_CompressionIdMask *(p. 573) mask with the default value for subscription.*

Typedefs

- typedef **DDS_UnsignedLong DDS_CompressionId_t**

  A four-byte signed integer that identifies a single compression algorithm.
- typedef **DDS_UnsignedLong DDS_CompressionIdMask**

  A four-byte signed integer that identifies a mask of the compression IDs that the algorithms enabled.

Variables

- const **DDS_CompressionId_t DDS_COMPRESSION_ID_ZLIB**

  Corresponds to the ID of the compression algorithm ZLIB. This is the only built-in compression algorithm that is supported when batching is enabled.
- const **DDS_CompressionId_t DDS_COMPRESSION_ID_BZIP2**

  Corresponds to the ID of the compression algorithm BZIP2.
- const **DDS_CompressionId_t DDS_COMPRESSION_ID_LZ4**

  Corresponds to the ID of the compression algorithm LZ4.

7.147.1 Detailed Description

<<***extension***>> *(p. 213)* Compression settings that can be applied to data that is sent and received by DataWriters and DataReaders.

7.147.2 Macro Definition Documentation

7.147.2.1 DDS_COMPRESSION_ID_MASK_NONE

```
#define DDS_COMPRESSION_ID_MASK_NONE (( DDS_CompressionIdMask) (0))
```

DDS_CompressionIdMask (p. 573) used to disable user-data compression for an endpoint.

7.147.2.2 DDS_COMPRESSION_ID_MASK_ALL

```
#define DDS_COMPRESSION_ID_MASK_ALL
```

Value:

```
((DDS_CompressionIdMask) \
        (DDS_COMPRESSION_ID_ZLIB_BIT | DDS_COMPRESSION_ID_BZIP2_BIT \
                | DDS_COMPRESSION_ID_LZ4_BIT))
```

DDS_CompressionIdMask (p. 573) mask with all the suported built-in compression algorithms enabled.

7.147.2.3 DDS_COMPRESSION_LEVEL_BEST_COMPRESSION

```
#define DDS_COMPRESSION_LEVEL_BEST_COMPRESSION (10)
```

The best compression ratio possible for a compression algorithm.

See also

DDS_CompressionSettings_t::writer_compression_level (p. 827)

7.147.2.4 DDS_COMPRESSION_LEVEL_BEST_SPEED

```
#define DDS_COMPRESSION_LEVEL_BEST_SPEED (1)
```

The best compression speed possible for a compression algorithm.

See also

DDS_CompressionSettings_t::writer_compression_level (p. 827)

7.147.2.5 DDS_COMPRESSION_LEVEL_DEFAULT

```
#define DDS_COMPRESSION_LEVEL_DEFAULT  DDS_COMPRESSION_LEVEL_BEST_COMPRESSION
```

Default level of compression.

See also

DDS_CompressionSettings_t::writer_compression_level (p. 827)

7.147.2.6 DDS_COMPRESSION_THRESHOLD_DEFAULT

```
#define DDS_COMPRESSION_THRESHOLD_DEFAULT 8192
```

Default threshold from which a serialized sample will be eligible to be compressed.

See also

DDS_CompressionSettings_t::writer_compression_threshold (p. 827)

This value has been decided after some performance tests, it's a prudent value that will ensure, in most cases with all of the algorithms, that we are going to compress the sample.

7.147.2.7 DDS_COMPRESSION_ID_MASK_PUBLICATION_DEFAULT

```
#define DDS_COMPRESSION_ID_MASK_PUBLICATION_DEFAULT  DDS_COMPRESSION_ID_MASK_NONE
```

DDS_CompressionIdMask (p. 573) mask with the default value for publication.

See also

DDS_CompressionSettings_t::compression_ids (p. 826)

DDS_COMPRESSION_ID_MASK_SUBSCRIPTION_DEFAULT (p. 572)

7.147.2.8 DDS_COMPRESSION_ID_MASK_SUBSCRIPTION_DEFAULT

```
#define DDS_COMPRESSION_ID_MASK_SUBSCRIPTION_DEFAULT
```

Value:

```
(DDS_COMPRESSION_ID_ZLIB_BIT | DDS_COMPRESSION_ID_BZIP2_BIT \
    | DDS_COMPRESSION_ID_LZ4_BIT)
```

DDS_CompressionIdMask (p. 573) mask with the default value for subscription.

See also

DDS_CompressionSettings_t::compression_ids (p. 826)

DDS_COMPRESSION_ID_MASK_PUBLICATION_DEFAULT (p. 572)

7.147.3 Typedef Documentation

7.147.3.1 DDS_CompressionId_t

```
typedef DDS_UnsignedLong DDS_CompressionId_t
```

A four-byte signed integer that identifies a single compression algorithm.

7.147.3.2 DDS_CompressionIdMask

```
typedef DDS_UnsignedLong DDS_CompressionIdMask
```

A four-byte signed integer that identifies a mask of the compression IDs that the algorithms enabled.

7.147.4 Variable Documentation

7.147.4.1 DDS_COMPRESSION_ID_ZLIB

```
const DDS_CompressionId_t DDS_COMPRESSION_ID_ZLIB [extern]
```

Corresponds to the ID of the compression algorithm ZLIB. This is the only built-in compression algorithm that is supported when batching is enabled.

7.147.4.2 DDS_COMPRESSION_ID_BZIP2

```
const DDS_CompressionId_t DDS_COMPRESSION_ID_BZIP2 [extern]
```

Corresponds to the ID of the compression algorithm BZIP2.

7.147.4.3 DDS_COMPRESSION_ID_LZ4

```
const DDS_CompressionId_t DDS_COMPRESSION_ID_LZ4 [extern]
```

Corresponds to the ID of the compression algorithm LZ4.

7.148 DATA_TAG

<<***spec***>> *(*p. *209)* Stores (name, value) pairs that can be used to determine access permissions.

Classes

- struct **DDS_Tag**
  Tags are name/value pair objects.
- struct **DDS_TagSeq**
  Declares IDL `sequence` *<* ***DDS_Tag*** *(*p. *1384)* *>*
- struct **DDS_DataTags**
  Definition of ***DDS_DataTagQosPolicy*** *(*p. *575).*
- class **DDSDataTagQosPolicyHelper**
  Policy helpers that facilitate management of the data tags in the input policy.

Typedefs

- typedef struct **DDS_DataTags** **DDS_DataTagQosPolicy**
  <<***spec***>> *(*p. *209) Stores (name, value) pairs that can be used to determine access permissions.*

Functions

- static **DDS_Long** **DDSDataTagQosPolicyHelper::get_number_of_tags** (**DDS_DataTagQosPolicy** &policy)
  Gets the number of data tags in the input policy.
- static **DDS_ReturnCode_t** **DDSDataTagQosPolicyHelper::assert_tag** (**DDS_DataTagQosPolicy** &policy, const char ∗name, const char ∗value)
  Asserts the tag identified by name in the input policy.
- static **DDS_ReturnCode_t** **DDSDataTagQosPolicyHelper::add_tag** (**DDS_DataTagQosPolicy** &policy, const char ∗name, const char ∗value)
  Adds a new tag to the input policy.
- static struct **DDS_Tag** ∗ **DDSDataTagQosPolicyHelper::lookup_tag** (**DDS_DataTagQosPolicy** &policy, const char ∗name)
  Searches by tag name for a tag in the input policy.
- static **DDS_ReturnCode_t** **DDSDataTagQosPolicyHelper::remove_tag** (**DDS_DataTagQosPolicy** &policy, const char ∗name)
  Removes a tag from the input policy.

Variables

- const char ∗const **DDS_DATATAG_QOS_POLICY_NAME**

  *Stringified human-readable name for **DDS_DataTagQosPolicy** (p. 575).*

7.148.1 Detailed Description

<<***spec***>> *(p. 209)* Stores (name, value) pairs that can be used to determine access permissions.

The **DDS_DataTagQosPolicy** (p. 575) can be used to associate a set of tags in the form of (name, value) pairs with a **DDSDataReader** (p. 1556) or **DDSDataWriter** (p. 1589). This is similar to the **DDS_PropertyQosPolicy** (p. 1260), except you cannot select whether or not a particular pair should be propagated (included in the built-in topic). Data tags are always propagated. The Access Control plugin may use the tags to determine publish and subscribe permissions.

7.148.2 Typedef Documentation

7.148.2.1 DDS_DataTagQosPolicy

`typedef struct` **`DDS_DataTags`** **`DDS_DataTagQosPolicy`**

<<***spec***>> *(p. 209)* Stores (name, value) pairs that can be used to determine access permissions.

Entity:

DDSDataReader (p. 1556) **DDSDataWriter** (p. 1589)

Properties:

RxO (p. 556) = N/A;
Changeable (p. 556) = **NO** (p. 556)

7.148.3 Usage

The DATA_TAG QoS policy can be used to associate a set of tags in the form of (name, value) pairs with a **DDSData↩ Reader** (p. 1556) or **DDSDataWriter** (p. 1589). This is similar to the **DDS_PropertyQosPolicy** (p. 1260), except for the following differences:

- Data tags are always propagated. You cannot select whether or not a particular pair should be propagated.
- Data tags are not exposed to API functions, such as **DDSDataWriter::get_matched_subscription_data** (p. 1600), that receive **DDS_PublicationBuiltinTopicData** (p. 1264) or **DDS_SubscriptionBuiltinTopicData** (p. 1370) as a parameter.
- Connext passes data tags to the Access Control Security Plugin, which may use them to decide whether to allow or deny the corresponding entities.

There are helper functions to facilitate working with data tags. See the **DATA_TAG** (p. 574) page.

7.148.4 Function Documentation

7.148.4.1 get_number_of_tags()

```
static  DDS_Long DDSDataTagQosPolicyHelper::get_number_of_tags (
              DDS_DataTagQosPolicy & policy )  [static]
```

Gets the number of data tags in the input policy.

Precondition

policy cannot be NULL.

Parameters

policy	<<***in***>> *(*p. *433)* Input policy.

Returns

Number of data tags.

7.148.4.2 assert_tag()

```
static  DDS_ReturnCode_t DDSDataTagQosPolicyHelper::assert_tag (
              DDS_DataTagQosPolicy & policy,
             const char * name,
             const char * value )  [static]
```

Asserts the tag identified by name in the input policy.

If the tag already exists, this function replaces its current value with the new one.

If the tag identified by name does not exist, this function adds it to the tag set.

This function increases the maximum number of elements of the policy sequence by 10 when this number is not enough to store the new tag.

Precondition

policy, name and value cannot be NULL.

Parameters

policy	<<***in***>> *(*p. *433)* Input policy.
name	<<***in***>> *(*p. *433)* Tag name.
value	<<***in***>> *(*p. *433)* Tag value.

Exceptions

One	of the **Standard Return Codes** (p. 532), **DDS_RETCODE_OUT_OF_RESOURCES** (p. 533)

7.148.4.3 add_tag()

```
static  DDS_ReturnCode_t DDSDataTagQosPolicyHelper::add_tag (
              DDS_DataTagQosPolicy & policy,
             const char * name,
             const char * value )  [static]
```

Adds a new tag to the input policy.

This function will allocate memory to store the (name, value) pair. The memory allocated is owned by RTI Connext.

If the maximum number of elements of the policy sequence is not enough to store the new tag, this function will increase it by 10.

If the tag already exists, the function will fail with **DDS_RETCODE_PRECONDITION_NOT_MET** (p. 532).

Precondition

policy, name and value cannot be NULL.

The tag is not in the policy.

Parameters

policy	<<***in***>> *(*p. *433)* Input policy.
name	<<***in***>> *(*p. *433)* Tag name.
value	<<***in***>> *(*p. *433)* Tag value.

Exceptions

One	of the **Standard Return Codes** (p. 532), **DDS_RETCODE_PRECONDITION_NOT_MET** (p. 532), **DDS_RETCODE_OUT_OF_RESOURCES** (p. 533)

7.148.4.4 lookup_tag()

```
static struct  DDS_Tag * DDSDataTagQosPolicyHelper::lookup_tag (
              DDS_DataTagQosPolicy & policy,
             const char * name )  [static]
```

Searches by tag name for a tag in the input policy.

Precondition

policy, name and value cannot be NULL.

Parameters

policy	<<***in***>> *(*p. *433)* Input policy.
name	<<***in***>> *(*p. *433)* Tag name.

Returns

The function returns the first tag with the given name. If such a tag does not exist, the function returns NULL.

7.148.4.5 remove_tag()

```
static  DDS_ReturnCode_t DDSDataTagQosPolicyHelper::remove_tag (
              DDS_DataTagQosPolicy & policy,
             const char * name )  [static]
```

Removes a tag from the input policy.

If the tag does not exist, the function fails with **DDS_RETCODE_PRECONDITION_NOT_MET** (p. 532).

Precondition

policy and name cannot be NULL.

The tag is in the policy.

Parameters

policy	<<***in***>> *(*p. *433)* Input policy.
name	<<***in***>> *(*p. *433)* Tag name.

Exceptions

One	of the **Standard Return Codes** (p. 532), **DDS_RETCODE_PRECONDITION_NOT_MET** (p. 532)

7.148.5 Variable Documentation

7.148.5.1 DDS_DATATAG_QOS_POLICY_NAME

```
const char* const DDS_DATATAG_QOS_POLICY_NAME  [extern]
```

Stringified human-readable name for **DDS_DataTagQosPolicy** (p. 575).

7.149 DATA_WRITER_PROTOCOL

<<***extension***>> *(p. 213)* Along with **DDS_WireProtocolQosPolicy** (p. 1511) and **DDS_DataReaderProtocolQos↩ Policy** (p. 843), this QoS policy configures the DDS on-the-network protocol (RTPS).

Classes

- struct **DDS_DataWriterProtocolQosPolicy**

  <<***extension***>> *(p. 213) Protocol that applies only to **DDSDataWriter** (p. 1589) instances.*

Variables

- const char *const **DDS_DATAWRITERPROTOCOL_QOS_POLICY_NAME**

  *Stringified human-readable name for **DDS_DataWriterProtocolQosPolicy** (p. 892).*

7.149.1 Detailed Description

<<***extension***>> *(p. 213)* Along with **DDS_WireProtocolQosPolicy** (p. 1511) and **DDS_DataReaderProtocolQos↩ Policy** (p. 843), this QoS policy configures the DDS on-the-network protocol (RTPS).

7.149.2 Variable Documentation

7.149.2.1 DDS_DATAWRITERPROTOCOL_QOS_POLICY_NAME

```
const char* const DDS_DATAWRITERPROTOCOL_QOS_POLICY_NAME  [extern]
```

Stringified human-readable name for **DDS_DataWriterProtocolQosPolicy** (p. 892).

7.150 DATA_WRITER_RESOURCE_LIMITS

<<***extension***>> *(p. 213)* Various settings that configure how a **DDSDataWriter** (p. 1589) allocates and uses physical memory for internal resources.

Classes

- struct **DDS_DataWriterResourceLimitsQosPolicy**

  <<***extension***>> *(p. 213) Various settings that configure how a **DDSDataWriter** (p. 1589) allocates and uses physical memory for internal resources.*

Enumerations

- enum **DDS_DataWriterResourceLimitsInstanceReplacementKind** {
  DDS_UNREGISTERED_INSTANCE_REPLACEMENT ,
  DDS_ALIVE_INSTANCE_REPLACEMENT ,
  DDS_DISPOSED_INSTANCE_REPLACEMENT ,
  DDS_ALIVE_THEN_DISPOSED_INSTANCE_REPLACEMENT ,
  DDS_DISPOSED_THEN_ALIVE_INSTANCE_REPLACEMENT ,
  DDS_ALIVE_OR_DISPOSED_INSTANCE_REPLACEMENT }

  Sets the kinds of instances that can be replaced when instance resource limits are reached.

Variables

- const char ∗const **DDS_DATAWRITERRESOURCELIMITS_QOS_POLICY_NAME**

  *Stringified human-readable name for **DDS_DataWriterResourceLimitsQosPolicy** (p. 917).*

7.150.1 Detailed Description

<<***extension***>> *(p. 213)* Various settings that configure how a **DDSDataWriter** (p. 1589) allocates and uses physical memory for internal resources.

7.150.2 Enumeration Type Documentation

7.150.2.1 DDS_DataWriterResourceLimitsInstanceReplacementKind

enum **DDS_DataWriterResourceLimitsInstanceReplacementKind**

Sets the kinds of instances that can be replaced when instance resource limits are reached.

When **DDS_ResourceLimitsQosPolicy::max_instances** (p. 1311) is reached, a **DDSDataWriter** (p. 1589) will try to make room for a new instance by attempting to reclaim an existing instance based on the instance replacement kind specified by **DDS_DataWriterResourceLimitsQosPolicy::instance_replacement** (p. 921).

Only instances whose states match the specified kinds are eligible to be replaced. In addition, an instance must have had all of its samples fully acknowledged for it to be considered replaceable.

For all kinds, a **DDSDataWriter** (p. 1589) will replace the oldest instance satisfying that kind. For example, when the kind is **DDS_UNREGISTERED_INSTANCE_REPLACEMENT** (p. 581), a **DDSDataWriter** (p. 1589) will remove the oldest, fully acknowledged, unregistered instance, if such an instance exists.

If no replaceable instance exists, the invoked function will either return with an appropriate out-of-resources return code, or in the case of a write, it may first block to wait for an instance to be acknowledged. Otherwise, the **DDSDataWriter** (p. 1589) will replace the old instance with the new instance, and invoke, if available, the **DDSDataWriterListener::on↩ _instance_replaced** (p. 1617) to notify the user about an instance being replaced.

A **DDSDataWriter** (p. 1589) checks for replaceable instances in the following order, stopping once a replaceable instance is found:

- If **DDS_DataWriterResourceLimitsQosPolicy::replace_empty_instances** (p. 922) is **DDS_BOOLEAN_TRUE** (p. 513), a **DDSDataWriter** (p. 1589) first tries replacing instances that have no samples. These empty instances can be unregistered, disposed, or alive.
- Next, a **DDSDataWriter** (p. 1589) tries replacing unregistered instances. Since an unregistered instance indicates that the **DDSDataWriter** (p. 1589) is done modifying it, unregistered instances are replaced before instances of any other state (alive, disposed). This is the same as the **DDS_UNREGISTERED_INSTANCE_REPLACEMENT** (p. 581) kind.
- Then, a **DDSDataWriter** (p. 1589) tries replacing what is specified by **DDS_DataWriterResourceLimitsQos↩ Policy::instance_replacement** (p. 921). With unregistered instances already checked, this leaves alive and disposed instances. When both alive and disposed instances may be replaced, the kind specifies whether the particular order matters (e.g., DISPOSED_THEN_ALIVE, ALIVE_THEN_DISPOSED) or not (ALIVE_OR_DISPOSED).

QoS:

DDS_DataWriterResourceLimitsQosPolicy (p. 917)

Enumerator

DDS_UNREGISTERED_INSTANCE_REPLACEMENT	Allows a **DDSDataWriter** (p. 1589) to reclaim unregistered acknowledged instances. By default, all instance replacement kinds first attempt to reclaim an unregistered, acknowledged instance. Used in **DDS_DataWriterResourceLimitsQosPolicy↩ ::instance_replacement** (p. 921) **[default]**

Enumerator

DDS_ALIVE_INSTANCE_REPLACEMENT	Allows a **DDSDataWriter** (p. 1589) to reclaim alive, acknowledged instances. When an unregistered, acknowledged instance is not available to reclaim, this kind allows a **DDSDataWriter** (p. 1589) to reclaim an alive, acknowledged instance, where an alive instance is a registered, non-disposed instance. The least recently registered or written alive instance will be reclaimed.
DDS_DISPOSED_INSTANCE_REPLACEMENT	Allows a **DDSDataWriter** (p. 1589) to reclaim disposed acknowledged instances. When an unregistered, acknowledged instance is not available to reclaim, this kind allows a **DDSDataWriter** (p. 1589) to reclaim a disposed, acknowledged instance. The least recently disposed instance will be reclaimed.
DDS_ALIVE_THEN_DISPOSED_INSTANCE_↩ REPLACEMENT	Allows a **DDSDataWriter** (p. 1589) first to reclaim an alive, acknowledged instance, and then, if necessary, a disposed, acknowledged instance. When an unregistered, acknowledged instance is not available to reclaim, this kind allows a **DDSDataWriter** (p. 1589) to first try reclaiming an alive, acknowledged instance. If no instance is reclaimable, then it tries reclaiming a disposed, acknowledged instance. The least recently used (i.e., registered, written, or disposed) instance will be reclaimed.
DDS_DISPOSED_THEN_ALIVE_INSTANCE_↩ REPLACEMENT	Allows a **DDSDataWriter** (p. 1589) first to reclaim a disposed, acknowledged instance, and then, if necessary, an alive, acknowledged instance. When an unregistered, acknowledged instance is not available to reclaim, this kind allows a **DDSDataWriter** (p. 1589) to first try reclaiming a disposed, acknowledged instance. If no instance is reclaimable, then it tries reclaiming an alive, acknowledged instance. The least recently used (i.e., disposed, registered, or written) instance will be reclaimed.
DDS_ALIVE_OR_DISPOSED_INSTANCE_↩ REPLACEMENT	Allows a **DDSDataWriter** (p. 1589) to reclaim a either an alive acknowledged instance or a disposed acknowledged instance. When an unregistered acknowledged instance is not available to reclaim, this kind allows a **DDSDataWriter** (p. 1589) to reclaim either an alive, acknowledged instance or a disposed, acknowledged instance. If both instance kinds are available to reclaim, the **DDSDataWriter** (p. 1589) will reclaim the least recently used (i.e. disposed, registered, or written) instance.

7.150.3 Variable Documentation

7.150.3.1 DDS_DATAWRITERRESOURCELIMITS_QOS_POLICY_NAME

```
const char* const DDS_DATAWRITERRESOURCELIMITS_QOS_POLICY_NAME  [extern]
```

Stringified human-readable name for **DDS_DataWriterResourceLimitsQosPolicy** (p. 917).

7.151 DATA_WRITER_TRANSFER_MODE

<<***extension***>> *(p. 213)* Specifies the DataWriter transfer mode QoS.

Classes

- struct **DDS_DataWriterShmemRefTransferModeSettings**
  Settings related to transferring data using shared memory references.
- struct **DDS_DataWriterTransferModeQosPolicy**
  <<***extension***>> *(p. 213) Qos related to transferring data*

Variables

- const char *const **DDS_DATAWRITERTRANSFERMODE_QOS_POLICY_NAME**
  *Stringified human-readable name for **DDS_DataWriterTransferModeQosPolicy** (p. 927).*

7.151.1 Detailed Description

<<***extension***>> *(p. 213)* Specifies the DataWriter transfer mode QoS.

7.151.2 Variable Documentation

7.151.2.1 DDS_DATAWRITERTRANSFERMODE_QOS_POLICY_NAME

```
const char* const DDS_DATAWRITERTRANSFERMODE_QOS_POLICY_NAME  [extern]
```

Stringified human-readable name for **DDS_DataWriterTransferModeQosPolicy** (p. 927).

7.152 DEADLINE

<<***spec***>> *(p. 209)* Expresses the maximum duration (deadline) within which an instance is expected to be updated.

Classes

- struct **DDS_DeadlineQosPolicy**

  *<<**spec**>> (p. 209) Expresses the maximum duration (deadline) within which an instance is expected to be updated.*

Variables

- const char *const **DDS_DEADLINE_QOS_POLICY_NAME**

  *Stringified human-readable name for **DDS_DeadlineQosPolicy** (p. 928).*

7.152.1 Detailed Description

<<**spec**>> *(p. 209)* Expresses the maximum duration (deadline) within which an instance is expected to be updated.

7.152.2 Variable Documentation

7.152.2.1 DDS_DEADLINE_QOS_POLICY_NAME

```
const char* const DDS_DEADLINE_QOS_POLICY_NAME  [extern]
```

Stringified human-readable name for **DDS_DeadlineQosPolicy** (p. 928).

7.153 DESTINATION_ORDER

<<**spec**>> *(p. 209)* Controls the criteria used to determine the logical order among changes made by **DDSPublisher** (p. 1818) entities to the same instance of data (i.e., matching **DDSTopic** (p. 1885) and key).

Classes

- struct **DDS_DestinationOrderQosPolicy**

  *<<**spec**>> (p. 209) Controls how the middleware will deal with data sent by multiple **DDSDataWriter** (p. 1589) entities for the same instance of data (i.e., same **DDSTopic** (p. 1885) and key).*

Enumerations

- enum **DDS_DestinationOrderQosPolicyKind** { **DDS_BY_RECEPTION_TIMESTAMP_DESTINATIONORDER_QOS** , **DDS_BY_SOURCE_TIMESTAMP_DESTINATIONORDER_QOS** }

  Kinds of destination order.
- enum **DDS_DestinationOrderQosPolicyScopeKind** { **DDS_INSTANCE_SCOPE_DESTINATIONORDER_QOS** , **DDS_TOPIC_SCOPE_DESTINATIONORDER_QOS** }

  Scope of source destination order.

Variables

- const char ∗const **DDS_DESTINATIONORDER_QOS_POLICY_NAME**

  *Stringified human-readable name for **DDS_DestinationOrderQosPolicy** (p. 931).*

7.153.1 Detailed Description

<<***spec***>> *(p. 209)* Controls the criteria used to determine the logical order among changes made by **DDSPublisher** (p. 1818) entities to the same instance of data (i.e., matching **DDSTopic** (p. 1885) and key).

7.153.2 Enumeration Type Documentation

7.153.2.1 DDS_DestinationOrderQosPolicyKind

`enum` **DDS_DestinationOrderQosPolicyKind**

Kinds of destination order.

QoS:

DDS_DestinationOrderQosPolicy (p. 931)

Enumerator

DDS_BY_RECEPTION_TIMESTAMP_↩ DESTINATIONORDER_QOS	**[default]** Indicates that data is ordered based on the reception time at each **DDSSubscriber** (p. 1860). Since each subscriber may receive the data at different times there is no guarantee that the changes will be seen in the same order. Consequently, it is possible for each subscriber to end up with a different final value for the data.
DDS_BY_SOURCE_TIMESTAMP_↩ DESTINATIONORDER_QOS	Indicates that data is ordered based on a time-stamp placed at the source (by RTI Connext or by the application). In any case this guarantees a consistent final value for the data in all subscribers. Note: If Batching is needed along with **DDS_BY_↩ SOURCE_TIMESTAMP_DESTINATIONORDER_QOS** (p. 585) and **DDS_INSTANCE_SCOPE_↩ DESTINATIONORDER_QOS** (p. 586), then the **DDS_BatchQosPolicy::source_timestamp_↩ resolution** (p. 812) and **DDS_BatchQosPolicy::thread_safe_write** (p. 813) setting of **DDS_BatchQosPolicy** (p. 810) should be set to **DDS_DURATION_ZERO** (p. 523) and **DDS_BOOLEAN_TRUE** (p. 513) respectively.

7.153.2.2 DDS_DestinationOrderQosPolicyScopeKind

enum **DDS_DestinationOrderQosPolicyScopeKind**

Scope of source destination order.

QoS:

DDS_DestinationOrderQosPolicy (p. 931)

Enumerator

DDS_INSTANCE_SCOPE_DESTINATIONORDER_QOS	**[default]** Indicates that data is ordered on a per instance basis if used along with **DDS_BY_SOURCE_TIMESTAMP_DESTINATIONORDER_QOS** (p. 585). The source timestamp of the current sample is compared to the source timestamp of the previously received sample for the same instance. The tolerance check is also applied per instance.
DDS_TOPIC_SCOPE_DESTINATIONORDER_QOS	Indicates that data is ordered on a per topic basis if used along with **DDS_BY_SOURCE_TIMESTAMP_DESTINATIONORDER_QOS** (p. 585). The source timestamp of the current sample is compared to the source timestamp of the previously received sample for the same topic. The tolerance check is also applied per topic.

7.153.3 Variable Documentation

7.153.3.1 DDS_DESTINATIONORDER_QOS_POLICY_NAME

const char* const DDS_DESTINATIONORDER_QOS_POLICY_NAME [extern]

Stringified human-readable name for **DDS_DestinationOrderQosPolicy** (p. 931).

7.154 DISCOVERY

<<***extension***>> *(p. 213)* Specifies the attributes required to discover participants in the domain.

Modules

- **NDDS_DISCOVERY_PEERS**

  *Environment variable or a file that specifies the default values of **DDS_DiscoveryQosPolicy::initial_peers** (p. 962) and **DDS_DiscoveryQosPolicy::multicast_receive_addresses** (p. 962) contained in the **DDS_DomainParticipantQos::discovery** (p. 974) qos policy.*

Classes

- struct **DDS_DiscoveryQosPolicy**

  *<<**extension**>> (p. 213) Configures the mechanism used by the middleware to automatically discover and connect with new remote applications.*

Variables

- const char *const **DDS_DISCOVERY_QOS_POLICY_NAME**

  *Stringified human-readable name for **DDS_DiscoveryQosPolicy** (p. 960).*

7.154.1 Detailed Description

<<**extension**>> *(p. 213)* Specifies the attributes required to discover participants in the domain.

7.154.2 Variable Documentation

7.154.2.1 DDS_DISCOVERY_QOS_POLICY_NAME

```
const char* const DDS_DISCOVERY_QOS_POLICY_NAME  [extern]
```

Stringified human-readable name for **DDS_DiscoveryQosPolicy** (p. 960).

7.155 DISCOVERY_CONFIG

<<**extension**>> *(p. 213)* Specifies the discovery configuration QoS.

Classes

- struct **DDS_BuiltinTopicReaderResourceLimits_t**

  *<<**extension**>> (p. 213) Built-in topic reader's resource limits.*
- struct **DDS_DiscoveryConfigQosPolicy**

  Settings for discovery configuration.

Macros

- #define **DDS_DISCOVERYCONFIG_BUILTIN_PLUGIN_MASK_NONE**

  No bits are set.
- #define **DDS_DISCOVERYCONFIG_BUILTIN_PLUGIN_MASK_DEFAULT**

  *The default value of **DDS_DiscoveryConfigQosPolicy::builtin_discovery_plugins** (p. 946).*
- #define **DDS_DISCOVERYCONFIG_BUILTIN_CHANNEL_MASK_NONE**

  *No bits are set, indicating that all channels that can be disabled, are disabled. Not all builtin channels can be disabled by the user. Only builtin channels represented by **DDS_DiscoveryConfigBuiltinChannelKind** (p. 592) are able to be enabled or disabled by the user.*
- #define **DDS_DISCOVERYCONFIG_BUILTIN_CHANNEL_MASK_DEFAULT**

  *The default value of **DDS_DiscoveryConfigQosPolicy::enabled_builtin_channels** (p. 946).*
- #define **DDS_DISCOVERYCONFIG_BUILTIN_CHANNEL_MASK_ALL**

  All bits are set, indicating that all channels are enabled.

Typedefs

- typedef **DDS_Long DDS_DiscoveryConfigBuiltinPluginKindMask**

  A bit-mask (list) of built-in discovery plugins.
- typedef **DDS_UnsignedLong DDS_DiscoveryConfigBuiltinChannelKindMask**

  A bit-mask (list) of built-in channels.

Enumerations

- enum **DDS_DiscoveryConfigBuiltinPluginKind** {
  DDS_DISCOVERYCONFIG_BUILTIN_SPDP ,
  DDS_DISCOVERYCONFIG_BUILTIN_SEDP ,
  DDS_DISCOVERYCONFIG_BUILTIN_SDP ,
  DDS_DISCOVERYCONFIG_BUILTIN_EDS = 0x0001 << 2 ,
  DDS_DISCOVERYCONFIG_BUILTIN_DPSE ,
  DDS_DISCOVERYCONFIG_BUILTIN_SPDP2 ,
  DDS_DISCOVERYCONFIG_BUILTIN_SDP2 }

  Built-in discovery plugins that can be used.
- enum **DDS_DiscoveryConfigBuiltinChannelKind** {
  DDS_DISCOVERYCONFIG_SERVICE_REQUEST_CHANNEL ,
  DDS_DISCOVERYCONFIG_TYPE_LOOKUP_SERVICE_REQUEST_CHANNEL ,
  DDS_DISCOVERYCONFIG_TYPE_LOOKUP_SERVICE_REPLY_CHANNEL ,
  DDS_DISCOVERYCONFIG_TYPE_LOOKUP_SERVICE_CHANNEL }

  Built-in channels that can be enabled.
- enum **DDS_RemoteParticipantPurgeKind** {
  DDS_LIVELINESS_BASED_REMOTE_PARTICIPANT_PURGE ,
  DDS_NO_REMOTE_PARTICIPANT_PURGE }

  Available behaviors for halting communication with remote participants (and their contained entities) with which discovery communication has been lost.

Variables

- const char *const **DDS_DISCOVERYCONFIG_QOS_POLICY_NAME**

  *Stringified human-readable name for **DDS_DiscoveryConfigQosPolicy** (p. 934).*

7.155.1 Detailed Description

<<***extension***>> *(p. 213)* Specifies the discovery configuration QoS.

7.155.2 Macro Definition Documentation

7.155.2.1 DDS_DISCOVERYCONFIG_BUILTIN_PLUGIN_MASK_NONE

```
#define DDS_DISCOVERYCONFIG_BUILTIN_PLUGIN_MASK_NONE
```

No bits are set.

See also

DDS_DiscoveryConfigBuiltinPluginKindMask (p. 590)

7.155.2.2 DDS_DISCOVERYCONFIG_BUILTIN_PLUGIN_MASK_DEFAULT

```
#define DDS_DISCOVERYCONFIG_BUILTIN_PLUGIN_MASK_DEFAULT
```

The default value of **DDS_DiscoveryConfigQosPolicy::builtin_discovery_plugins** (p. 946).

See also

DDS_DiscoveryConfigBuiltinPluginKindMask (p. 590)

7.155.2.3 DDS_DISCOVERYCONFIG_BUILTIN_CHANNEL_MASK_NONE

```
#define DDS_DISCOVERYCONFIG_BUILTIN_CHANNEL_MASK_NONE
```

No bits are set, indicating that all channels that can be disabled, are disabled. Not all builtin channels can be disabled by the user. Only builtin channels represented by **DDS_DiscoveryConfigBuiltinChannelKind** (p. 592) are able to be enabled or disabled by the user.

See also

DDS_DiscoveryConfigBuiltinChannelKindMask (p. 590)

7.155.2.4 DDS_DISCOVERYCONFIG_BUILTIN_CHANNEL_MASK_DEFAULT

```
#define DDS_DISCOVERYCONFIG_BUILTIN_CHANNEL_MASK_DEFAULT
```

The default value of **DDS_DiscoveryConfigQosPolicy::enabled_builtin_channels** (p. 946).

See also

DDS_DiscoveryConfigBuiltinChannelKindMask (p. 590)

7.155.2.5 DDS_DISCOVERYCONFIG_BUILTIN_CHANNEL_MASK_ALL

```
#define DDS_DISCOVERYCONFIG_BUILTIN_CHANNEL_MASK_ALL
```

All bits are set, indicating that all channels are enabled.

See also

DDS_DiscoveryConfigBuiltinChannelKindMask (p. 590)

7.155.3 Typedef Documentation

7.155.3.1 DDS_DiscoveryConfigBuiltinPluginKindMask

```
typedef DDS_Long DDS_DiscoveryConfigBuiltinPluginKindMask
```

A bit-mask (list) of built-in discovery plugins.

The bit-mask is an efficient and compact representation of a fixed-length list of **DDS_DiscoveryConfigBuiltinPlugin↩ Kind** (p. 591) values.

QoS:

DDS_DiscoveryConfigQosPolicy (p. 934)

7.155.3.2 DDS_DiscoveryConfigBuiltinChannelKindMask

```
typedef DDS_UnsignedLong DDS_DiscoveryConfigBuiltinChannelKindMask
```

A bit-mask (list) of built-in channels.

The bit-mask is an efficient and compact representation of a fixed-length list of **DDS_DiscoveryConfigBuiltin↩ ChannelKind** (p. 592) values.

QoS:

DDS_DiscoveryConfigQosPolicy (p. 934)

7.155.4 Enumeration Type Documentation

7.155.4.1 DDS_DiscoveryConfigBuiltinPluginKind

```
enum DDS_DiscoveryConfigBuiltinPluginKind
```

Built-in discovery plugins that can be used.

See also

DDS_DiscoveryConfigBuiltinPluginKindMask (p. 590)

Enumerator

DDS_DISCOVERYCONFIG_BUILTIN_SPDP	Simple Participant Discovery Protocol. Enables the first phase of the Simple Discovery Protocol (SDP), in which DomainParticipant's details are communicated to all other DomainParticipants in the same DDS domain by sending participant declaration messages, also known as participant DATA submessages or participant announcements.
DDS_DISCOVERYCONFIG_BUILTIN_SEDP	Simple Endpoint Discovery Protocol. Enables the second phase of the Simple Discovery Protocol (SDP), in which the information (GUID, QoS, etc.) about your application's DataReaders and DataWriters is exchanged by sending publication/subscription declarations in DATA messages, also known as publication DATAs and subscription DATAs.
DDS_DISCOVERYCONFIG_BUILTIN_SDP	**[default]** Simple discovery plugin. It is equivalent to SPDP + SEDP.

Enumerator

DDS_DISCOVERYCONFIG_BUILTIN_DPSE	Dynamic Participant discovery, Static Endpoint discovery. Enables static endpoint discovery for a DomainParticipant. In this type of discovery, information from remote endpoints is extracted from a local DDS-XML file instead of being received over the network, reducing the number of exchanged packets and consequently reducing bandwidth consumption used for discovery. Using this value in **DDS_DiscoveryConfigBuiltinPluginKindMask** (p. 590) requires the 'librtilbedisc' library (included in the RTI Connext Professional bundles) to be reachable (PATH, LD_LIBRARY_PATH or DYLD_LIBRARY_PATH).
DDS_DISCOVERYCONFIG_BUILTIN_SPDP2	Simple Participant Discovery Protocol 2.0 (SPDP2) Enables the Simple Participant Discovery Protocol 2.0, in which a DomainParticipant's details are communicated to all other DomainParticipants in the same DDS domain by sending participant bootstrap messages. These bootstrap messages contain only a subset of the information in the Simple Participant Discovery Protocol (SPDP) participant announcements that is required to match two participants and bootstrap the system. The DomainParticpant's full configuration is then sent reliably with participant configuration announcements. Two DomainParticipants that use SPDP2 will maintain liveliness using liveliness participant messages.
DDS_DISCOVERYCONFIG_BUILTIN_SDP2	Simple discovery plugin 2.0. It is equivalent to SPDP2 + SEDP.

7.155.4.2 DDS_DiscoveryConfigBuiltinChannelKind

`enum` **`DDS_DiscoveryConfigBuiltinChannelKind`**

Built-in channels that can be enabled.

See also

DDS_DiscoveryConfigBuiltinChannelKindMask (p. 590)

Enumerator

DDS_DISCOVERYCONFIG_SERVICE_REQUEST_↩ CHANNEL	**[default]** Built-in ServiceRequest channel. Enables the ServiceRequest channel, which is required by the TopicQuery and locator reachability features. Disabling the ServiceRequest channel reduces resource consumption including network bandwidth, CPU utilization, and memory.
DDS_DISCOVERYCONFIG_TYPE_LOOKUP_↩ SERVICE_CHANNEL	**[default]** Built-in Type Lookup Service channel. Enables both the Type Lookup Service Request channel and Type Lookup Service Reply channel.

7.155.4.3 DDS_RemoteParticipantPurgeKind

enum **DDS_RemoteParticipantPurgeKind**

Available behaviors for halting communication with remote participants (and their contained entities) with which discovery communication has been lost.

When discovery communication with a remote participant has been lost, the local participant must make a decision about whether to continue attempting to communicate with that participant and its contained entities. This "kind" is used to select the desired behavior.

This "kind" does not pertain to the situation in which a remote participant has been gracefully deleted and notification of that deletion have been successfully received by its peers. In that case, the local participant will immediately stop attempting to communicate with those entities and will remove the associated remote entity records from its internal database.

See also

DDS_DiscoveryConfigQosPolicy::remote_participant_purge_kind (p. 938)

Enumerator

DDS_LIVELINESS_BASED_REMOTE_↩ PARTICIPANT_PURGE	**[default]** Maintain knowledge of the remote participant for as long as it maintains its liveliness contract. A participant will continue attempting communication with its peers, even if discovery communication with them is lost, as long as the remote participants maintain their liveliness. If both discovery communication and participant liveliness are lost, however, the local participant will remove all records of the remote participant and its contained endpoints, and no further data communication with them will occur until and unless they are rediscovered. The liveliness contract a participant promises to its peers – its "liveliness lease duration" – is specified in its **DDS_DiscoveryConfigQosPolicy::participant_↩ liveliness_lease_duration** (p. 937) QoS field. It maintains that contract by writing data to those other participants with a writer that has a **DDS_LivelinessQosPolicyKind** (p. 607) of **DDS_AUTOMATIC_LIVELINESS_QOS** (p. 608) or **DDS_MANUAL_BY_PARTICIPANT_LIVELINESS_↩ QOS** (p. 608) and by asserting itself (at the **DDS_DiscoveryConfigQosPolicy::participant_↩ liveliness_assert_period** (p. 937)) over the Simple Discovery Protocol.

Enumerator

DDS_NO_REMOTE_PARTICIPANT_PURGE	Never "forget" a remote participant with which discovery communication has been lost. If a participant with this behavior loses discovery communication with a remote participant, it will nevertheless remember that remote participant and its endpoints and continue attempting to communicate with them indefinitely. This value has consequences for a participant's resource usage. If discovery communication with a remote participant is lost, but the same participant is later rediscovered, any relevant records that remain in the database will be reused. However, if it is not rediscovered, the records will continue to take up space in the database for as long as the local participant remains in existence.

7.155.5 Variable Documentation

7.155.5.1 DDS_DISCOVERYCONFIG_QOS_POLICY_NAME

```
const char* const DDS_DISCOVERYCONFIG_QOS_POLICY_NAME  [extern]
```

Stringified human-readable name for **DDS_DiscoveryConfigQosPolicy** (p. 934).

7.156 DOMAIN_PARTICIPANT_RESOURCE_LIMITS

<<***extension***>> *(p. 213)* Various settings that configure how a **DDSDomainParticipant** (p. 1619) allocates and uses physical memory for internal resources, including the maximum sizes of various properties.

Classes

- struct **DDS_AllocationSettings_t**

  Resource allocation settings.
- struct **DDS_DomainParticipantResourceLimitsQosPolicy**

  <<***extension***>> *(p. 213) Various settings that configure how a **DDSDomainParticipant** (p. 1619) allocates and uses physical memory for internal resources, including the maximum sizes of various properties.*

Enumerations

- enum **DDS_DomainParticipantResourceLimitsIgnoredEntityReplacementKind** { **DDS_NO_REPLACEMENT_IGNORED_ENTITY_REPLACEMENT** , **DDS_NOT_ALIVE_FIRST_IGNORED_ENTITY_REPLACEMENT** }

  Available replacement policies for the ignored entities.

Variables

- const **DDS_Long DDS_AUTO_COUNT**

  A special value indicating a quantity that is derived from another QoS value.
- const char ∗const **DDS_DOMAINPARTICIPANTRESOURCELIMITS_QOS_POLICY_NAME**

  *Stringified human-readable name for **DDS_DomainParticipantResourceLimitsQosPolicy** (p. 977).*

7.156.1 Detailed Description

<<***extension***>> *(p. 213)* Various settings that configure how a **DDSDomainParticipant** (p. 1619) allocates and uses physical memory for internal resources, including the maximum sizes of various properties.

7.156.2 Enumeration Type Documentation

7.156.2.1 DDS_DomainParticipantResourceLimitsIgnoredEntityReplacementKind

`enum` **`DDS_DomainParticipantResourceLimitsIgnoredEntityReplacementKind`**

Available replacement policies for the ignored entities.

QoS:

DDS_DomainParticipantResourceLimitsQosPolicy (p. 977)

Enumerator

DDS_NO_REPLACEMENT_IGNORED_ENTITY_↩ REPLACEMENT	Default value for ignored_entity_replacement_kind, no replacement is done, the ignore will fail if the ignored_entity table is full.
DDS_NOT_ALIVE_FIRST_IGNORED_ENTITY_↩ REPLACEMENT	If the ignored_entity table is full and if there is at least one ignored participant in the table, the participant record that has been not updated for the longest time will be replaced. Note that if the table is full and there are no participant records to replace, the ignore will fail.

7.156.3 Variable Documentation

7.156.3.1 DDS_AUTO_COUNT

```
const  DDS_Long DDS_AUTO_COUNT  [extern]
```

A special value indicating a quantity that is derived from another QoS value.

7.156.3.2 DDS_DOMAINPARTICIPANTRESOURCELIMITS_QOS_POLICY_NAME

```
const char* const DDS_DOMAINPARTICIPANTRESOURCELIMITS_QOS_POLICY_NAME  [extern]
```

Stringified human-readable name for **DDS_DomainParticipantResourceLimitsQosPolicy** (p. 977).

7.157 DURABILITY

<<***spec***>> *(p. 209)* This QoS policy specifies whether or not RTI Connext will store and deliver previously published data samples to new **DDSDataReader** (p. 1556) entities that join the network later.

Classes

- struct **DDS_PersistentStorageSettings**

  Configures durable writer history and durable reader state.
- struct **DDS_DurabilityQosPolicy**

  *<<**spec**>> (p. 209) This QoS policy specifies whether or not RTI Connext will store and deliver previously published data samples to new **DDSDataReader** (p. 1556) entities that join the network later.*

Enumerations

- enum **DDS_PersistentJournalKind** {
  DDS_DELETE_PERSISTENT_JOURNAL ,
  DDS_TRUNCATE_PERSISTENT_JOURNAL ,
  DDS_PERSIST_PERSISTENT_JOURNAL ,
  DDS_MEMORY_PERSISTENT_JOURNAL ,
  DDS_WAL_PERSISTENT_JOURNAL ,
  DDS_OFF_PERSISTENT_JOURNAL }

  Sets the journal mode of the persistent storage.
- enum **DDS_PersistentSynchronizationKind** {
  DDS_NORMAL_PERSISTENT_SYNCHRONIZATION ,
  DDS_FULL_PERSISTENT_SYNCHRONIZATION ,
  DDS_OFF_PERSISTENT_SYNCHRONIZATION }

  Determines the level of synchronization with the physical disk.
- enum **DDS_DurabilityQosPolicyKind** {
  DDS_VOLATILE_DURABILITY_QOS ,
  DDS_TRANSIENT_LOCAL_DURABILITY_QOS ,
  DDS_TRANSIENT_DURABILITY_QOS ,
  DDS_PERSISTENT_DURABILITY_QOS }

  Kinds of durability.

Variables

- const char ∗const **DDS_DURABILITY_QOS_POLICY_NAME**

  *Stringified human-readable name for **DDS_DurabilityQosPolicy** (p. 1005).*
- const **DDS_Long** **DDS_AUTO_WRITER_DEPTH**

  *A special value used as the default value for **DDS_DurabilityQosPolicy::writer_depth** (p. 1008).*

7.157.1 Detailed Description

<<***spec***>> *(p. 209)* This QoS policy specifies whether or not RTI Connext will store and deliver previously published data samples to new **DDSDataReader** (p. 1556) entities that join the network later.

7.157.2 Enumeration Type Documentation

7.157.2.1 DDS_PersistentJournalKind

enum **DDS_PersistentJournalKind**

Sets the journal mode of the persistent storage.

The rollback journal is used in SQLite to store the state of the persistent storage before a transaction is committed.

QoS:

DDS_DurabilityQosPolicy (p. 1005)

Enumerator

DDS_DELETE_PERSISTENT_JOURNAL	Deletes the rollback journal at the conclusion of each transaction.
DDS_TRUNCATE_PERSISTENT_JOURNAL	Commits transactions by truncating the rollback journal to zero-length instead of deleting it.
DDS_PERSIST_PERSISTENT_JOURNAL	Prevents the rollback journal from being deleted at the end of each transaction. Instead, the header of the journal is overwritten with zeros.
DDS_MEMORY_PERSISTENT_JOURNAL	Stores the rollback journal in volatile RAM. This saves disk I/O.
DDS_WAL_PERSISTENT_JOURNAL	Uses a write-ahead log instead of a rollback journal to implement transactions.
DDS_OFF_PERSISTENT_JOURNAL	Completely disables the rollback journal. If the application crashes in the middle of a transaction when the OFF journaling mode is set, the persistent storage will very likely be corrupted.

7.157.2.2 DDS_PersistentSynchronizationKind

enum **DDS_PersistentSynchronizationKind**

Determines the level of synchronization with the physical disk.

QoS:

DDS_DurabilityQosPolicy (p. 1005)

Enumerator

DDS_NORMAL_PERSISTENT_SYNCHRONIZATION	Data (e.g., new sample) is written to disk at critical moments.
DDS_FULL_PERSISTENT_SYNCHRONIZATION	Data (e.g., new sample) is written to physical disk immediately.
DDS_OFF_PERSISTENT_SYNCHRONIZATION	No synchronization is enforced. Data will be written to physical disk when the operating system flushes its buffers.

7.157.2.3 DDS_DurabilityQosPolicyKind

enum **DDS_DurabilityQosPolicyKind**

Kinds of durability.

QoS:

DDS_DurabilityQosPolicy (p. 1005)

Enumerator

DDS_VOLATILE_DURABILITY_QOS	**[default]** RTI Connext does not need to keep any samples of data instances on behalf of any **DDSDataReader** (p. 1556) that is unknown by the **DDSDataWriter** (p. 1589) at the time the instance is written. In other words, RTI Connext will only attempt to provide the data to existing subscribers. This option does not require RTI Persistence Service.

Enumerator

DDS_TRANSIENT_LOCAL_DURABILITY_QOS	RTI Connext will attempt to keep some samples so that they can be delivered to any potential late-joining **DDSDataReader** (p. 1556). Which particular samples are kept depends on other QoS such as **DDS_HistoryQosPolicy** (p. 1162) and **DDS_ResourceLimitsQosPolicy** (p. 1308). RTI Connext is only required to keep the data in memory of the **DDSDataWriter** (p. 1589) that wrote the data. Data is not required to survive the **DDSDataWriter** (p. 1589). For this setting to be effective, you must also set the **DDS_ReliabilityQosPolicy::kind** (p. 1298) to **DDS_RELIABLE_RELIABILITY_QOS** (p. 633). This option does not require RTI Persistence Service.
DDS_TRANSIENT_DURABILITY_QOS	RTI Connext will attempt to keep some samples so that they can be delivered to any potential late-joining **DDSDataReader** (p. 1556). Which particular samples are kept depends on other QoS such as **DDS_HistoryQosPolicy** (p. 1162) and **DDS_ResourceLimitsQosPolicy** (p. 1308). RTI Connext is only required to keep the data in memory and not in permanent storage. Data is not tied to the lifecycle of the **DDSDataWriter** (p. 1589). Data will survive the **DDSDataWriter** (p. 1589). This option requires RTI Persistence Service.
DDS_PERSISTENT_DURABILITY_QOS	Data is kept on permanent storage, so that they can outlive a system session. This option requires RTI Persistence Service.

7.157.3 Variable Documentation

7.157.3.1 DDS_DURABILITY_QOS_POLICY_NAME

```
const char* const DDS_DURABILITY_QOS_POLICY_NAME  [extern]
```

Stringified human-readable name for **DDS_DurabilityQosPolicy** (p. 1005).

7.157.3.2 DDS_AUTO_WRITER_DEPTH

```
const  DDS_Long DDS_AUTO_WRITER_DEPTH  [extern]
```

A special value used as the default value for **DDS_DurabilityQosPolicy::writer_depth** (p. 1008).

This values resolves to the following:

- **DDS_HistoryQosPolicy::depth** (p. 1164) if the history kind is **DDS_KEEP_LAST_HISTORY_QOS** (p. 604).
- **DDS_ResourceLimitsQosPolicy::max_samples_per_instance** (p. 1311) in the **DDS_ResourceLimitsQos↩ Policy** (p. 1308) if the history kind is **DDS_KEEP_ALL_HISTORY_QOS** (p. 604).

7.158 DURABILITY_SERVICE

<<***spec***>> *(*p. *209)* Various settings to configure the external *RTI Persistence Service* used by RTI Connext for DataWriters with a **DDS_DurabilityQosPolicy** (p. 1005) setting of **DDS_PERSISTENT_DURABILITY_QOS** (p. 599) or **DDS_TRANSIENT_DURABILITY_QOS** (p. 599).

Classes

- struct **DDS_DurabilityServiceQosPolicy**

  <<***spec***>> (p. *209) Various settings to configure the external RTI Persistence Service used by RTI Connext for Data↩ Writers with a **DDS_DurabilityQosPolicy** (p. 1005) setting of **DDS_PERSISTENT_DURABILITY_QOS** (p. 599) or **DDS↩ _TRANSIENT_DURABILITY_QOS** (p. 599).*

Variables

- const char ∗const **DDS_DURABILITYSERVICE_QOS_POLICY_NAME**

  *Stringified human-readable name for **DDS_DurabilityServiceQosPolicy** (p. 1009).*

7.158.1 Detailed Description

<<***spec***>> *(*p. *209)* Various settings to configure the external *RTI Persistence Service* used by RTI Connext for DataWriters with a **DDS_DurabilityQosPolicy** (p. 1005) setting of **DDS_PERSISTENT_DURABILITY_QOS** (p. 599) or **DDS_TRANSIENT_DURABILITY_QOS** (p. 599).

7.158.2 Variable Documentation

7.158.2.1 DDS_DURABILITYSERVICE_QOS_POLICY_NAME

```
const char* const DDS_DURABILITYSERVICE_QOS_POLICY_NAME  [extern]
```

Stringified human-readable name for **DDS_DurabilityServiceQosPolicy** (p. 1009).

7.159 ENTITY_FACTORY

<<***spec***>> *(*p. *209)* A QoS policy for all **DDSEntity** (p. 1731) types that can act as factories for one or more other **DDSEntity** (p. 1731) types.

Classes

- struct **DDS_EntityFactoryQosPolicy**

  *<<**spec**>> (p. 209) A QoS policy for all **DDSEntity** (p. 1731) types that can act as factories for one or more other **DDSEntity** (p. 1731) types.*

Variables

- const char *const **DDS_ENTITYFACTORY_QOS_POLICY_NAME**

  *Stringified human-readable name for **DDS_EntityFactoryQosPolicy** (p. 1145).*

7.159.1 Detailed Description

*<<**spec**>> (p. 209)* A QoS policy for all **DDSEntity** (p. 1731) types that can act as factories for one or more other **DDSEntity** (p. 1731) types.

7.159.2 Variable Documentation

7.159.2.1 DDS_ENTITYFACTORY_QOS_POLICY_NAME

```
const char* const DDS_ENTITYFACTORY_QOS_POLICY_NAME  [extern]
```

Stringified human-readable name for **DDS_EntityFactoryQosPolicy** (p. 1145).

7.160 ENTITY_NAME

*<<**extension**>> (p. 213)* Assigns a name to a **DDSDomainParticipant** (p. 1619), **DDSPublisher** (p. 1818), **DDSSubscriber** (p. 1860), **DDSDataWriter** (p. 1589) or **DDSDataReader** (p. 1556). Except for **DDSPublisher** (p. 1818) and **DDSSubscriber** (p. 1860), these names are visible during the discovery process and in RTI tools to help you visualize and debug your system.

Classes

- struct **DDS_EntityNameQosPolicy**

  *<<**extension**>> (p. 213) Assigns a name and a role name to a **DDSDomainParticipant** (p. 1619), **DDSPublisher** (p. 1818), **DDSSubscriber** (p. 1860), **DDSDataWriter** (p. 1589) or **DDSDataReader** (p. 1556). Except for **DDSPublisher** (p. 1818) and **DDSSubscriber** (p. 1860), these names will be visible during the discovery process and in RTI tools to help you visualize and debug your system.*

Variables

- const char ∗const **DDS_ENTITYNAME_QOS_POLICY_NAME**

  *Stringified human-readable name for **DDS_EntityNameQosPolicy** (p. 1147).*

7.160.1 Detailed Description

<<***extension***>> *(p. 213)* Assigns a name to a **DDSDomainParticipant** (p. 1619), **DDSPublisher** (p. 1818), **DDSSubscriber** (p. 1860), **DDSDataWriter** (p. 1589) or **DDSDataReader** (p. 1556). Except for **DDSPublisher** (p. 1818) and **DDSSubscriber** (p. 1860), these names are visible during the discovery process and in RTI tools to help you visualize and debug your system.

7.160.2 Variable Documentation

7.160.2.1 DDS_ENTITYNAME_QOS_POLICY_NAME

```
const char* const DDS_ENTITYNAME_QOS_POLICY_NAME  [extern]
```

Stringified human-readable name for **DDS_EntityNameQosPolicy** (p. 1147).

7.161 EVENT

<<***extension***>> *(p. 213)* Configures the internal thread in a DomainParticipant that handles timed events.

Classes

- struct **DDS_EventQosPolicy**

  <<***extension***>> *(p. 213) Settings for event.*

Variables

- const char ∗const **DDS_EVENT_QOS_POLICY_NAME**

  *Stringified human-readable name for **DDS_EventQosPolicy** (p. 1150).*

7.161.1 Detailed Description

<<***extension***>> *(p. 213)* Configures the internal thread in a DomainParticipant that handles timed events.

7.161.2 Variable Documentation

7.161.2.1 DDS_EVENT_QOS_POLICY_NAME

```
const char* const DDS_EVENT_QOS_POLICY_NAME  [extern]
```

Stringified human-readable name for **DDS_EventQosPolicy** (p. 1150).

7.162 HISTORY

<<***spec***>> *(p. 209)* Specifies the behavior of RTI Connext in the case where the value of an instance changes (one or more times) before it can be successfully communicated to one or more existing subscribers.

Classes

- struct **DDS_HistoryQosPolicy**

  <<***spec***>> (p. *209) Specifies the behavior of RTI Connext in the case where the value of a sample changes (one or more times) before it can be successfully communicated to one or more existing subscribers.*

Enumerations

- enum **DDS_HistoryQosPolicyKind** {
  DDS_KEEP_LAST_HISTORY_QOS ,
  DDS_KEEP_ALL_HISTORY_QOS }

  Kinds of history.

Variables

- const char ∗const **DDS_HISTORY_QOS_POLICY_NAME**

  *Stringified human-readable name for **DDS_HistoryQosPolicy** (p. 1162).*

7.162.1 Detailed Description

<<***spec***>> *(p. 209)* Specifies the behavior of RTI Connext in the case where the value of an instance changes (one or more times) before it can be successfully communicated to one or more existing subscribers.

7.162.2 Enumeration Type Documentation

7.162.2.1 DDS_HistoryQosPolicyKind

```
enum DDS_HistoryQosPolicyKind
```

Kinds of history.

QoS:

DDS_HistoryQosPolicy (p. 1162)

Enumerator

DDS_KEEP_LAST_HISTORY_QOS	**[default]** Keep the last `depth` samples. On the publishing side, RTI Connext will only attempt to keep the most recent `depth` samples of each instance of data (identified by its key) managed by the **DDSDataWriter** (p. 1589). Invalid samples representing a disposal or unregistration of an instance count towards the depth and may replace other DDS samples currently in the DataWriter queue for the same instance. On the subscribing side, the **DDSDataReader** (p. 1556) will only attempt to keep the most recent `depth` samples received for each instance (identified by its key) until the application takes them via the **DDSDataReader** (p. 1556) 's **take()** operation. Invalid samples representing a disposal or unregistration of an instance do not count towards the history depth and will not replace other DDS samples currently in the DataReader queue for the same instance.
DDS_KEEP_ALL_HISTORY_QOS	Keep *all* the samples. On the publishing side, RTI Connext will attempt to keep all samples (representing each value written) of each instance of data (identified by its key) managed by the **DDSDataWriter** (p. 1589) until they can be delivered to all subscribers. On the subscribing side, RTI Connext will attempt to keep all samples of each instance of data (identified by its key) managed by the **DDSDataReader** (p. 1556). These samples are kept until the application takes them from RTI Connext via the **take()** operation.

7.162.3 Variable Documentation

7.162.3.1 DDS_HISTORY_QOS_POLICY_NAME

```
const char* const DDS_HISTORY_QOS_POLICY_NAME  [extern]
```

Stringified human-readable name for **DDS_HistoryQosPolicy** (p. 1162).

7.163 GROUP_DATA

Attaches a buffer of opaque data that is distributed by means of **Built-in Topics** (p. 61) during discovery.

Classes

- struct **DDS_GroupDataQosPolicy**

  *Attaches a buffer of opaque data that is distributed by means of **Built-in Topics** (p. 61) during discovery.*

Variables

- const char ∗const **DDS_GROUPDATA_QOS_POLICY_NAME**

  *Stringified human-readable name for **DDS_GroupDataQosPolicy** (p. 1160).*

7.163.1 Detailed Description

Attaches a buffer of opaque data that is distributed by means of **Built-in Topics** (p. 61) during discovery.

7.163.2 Variable Documentation

7.163.2.1 DDS_GROUPDATA_QOS_POLICY_NAME

```
const char* const DDS_GROUPDATA_QOS_POLICY_NAME  [extern]
```

Stringified human-readable name for **DDS_GroupDataQosPolicy** (p. 1160).

7.164 LATENCY_BUDGET

<<***spec***>> *(p. 209)* Provides a hint as to the maximum acceptable delay from the time the data is written to the time it is received by the subscribing applications.

Classes

- struct **DDS_LatencyBudgetQosPolicy**

  *<<**spec**>> (p. 209) Provides a hint as to the maximum acceptable delay from the time the data is written to the time it is received by the subscribing applications.*

Variables

- const char ∗const **DDS_LATENCYBUDGET_QOS_POLICY_NAME**

  *Stringified human-readable name for **DDS_LatencyBudgetQosPolicy** (p. 1173).*

7.164.1 Detailed Description

<<***spec***>> *(p. 209)* Provides a hint as to the maximum acceptable delay from the time the data is written to the time it is received by the subscribing applications.

7.164.2 Variable Documentation

7.164.2.1 DDS_LATENCYBUDGET_QOS_POLICY_NAME

```
const char* const DDS_LATENCYBUDGET_QOS_POLICY_NAME  [extern]
```

Stringified human-readable name for **DDS_LatencyBudgetQosPolicy** (p. 1173).

7.165 LIFESPAN

<<***spec***>> *(p. 209)* Specifies how long the data written by the **DDSDataWriter** (p. 1589) is considered valid.

Classes

- struct **DDS_LifespanQosPolicy**
  <<***spec***>> *(p. 209) Specifies how long the data written by the **DDSDataWriter** (p. 1589) is considered valid.*

Variables

- const char *const **DDS_LIFESPAN_QOS_POLICY_NAME**
  *Stringified human-readable name for **DDS_LifespanQosPolicy** (p. 1175).*

7.165.1 Detailed Description

<<***spec***>> *(p. 209)* Specifies how long the data written by the **DDSDataWriter** (p. 1589) is considered valid.

7.165.2 Variable Documentation

7.165.2.1 DDS_LIFESPAN_QOS_POLICY_NAME

```
const char* const DDS_LIFESPAN_QOS_POLICY_NAME  [extern]
```

Stringified human-readable name for **DDS_LifespanQosPolicy** (p. 1175).

7.166 LIVELINESS

<<***spec***>> *(p. 209)* Specifies and configures the mechanism that allows **DDSDataReader** (p. 1556) entities to detect when **DDSDataWriter** (p. 1589) entities become disconnected or "dead."

Classes

- struct **DDS_LivelinessQosPolicy**

  *Specifies and configures the mechanism that allows **DDSDataReader** (p. 1556) entities to detect when **DDSDataWriter** (p. 1589) entities become disconnected or "dead.".*

Enumerations

- enum **DDS_LivelinessQosPolicyKind** {
  DDS_AUTOMATIC_LIVELINESS_QOS ,
  DDS_MANUAL_BY_PARTICIPANT_LIVELINESS_QOS ,
  DDS_MANUAL_BY_TOPIC_LIVELINESS_QOS }

  Kinds of liveliness.

Variables

- const char ∗const **DDS_LIVELINESS_QOS_POLICY_NAME**

  *Stringified human-readable name for **DDS_LivelinessQosPolicy** (p. 1180).*

7.166.1 Detailed Description

<<***spec***>> *(p. 209)* Specifies and configures the mechanism that allows **DDSDataReader** (p. 1556) entities to detect when **DDSDataWriter** (p. 1589) entities become disconnected or "dead."

7.166.2 Enumeration Type Documentation

7.166.2.1 DDS_LivelinessQosPolicyKind

`enum` **`DDS_LivelinessQosPolicyKind`**

Kinds of liveliness.

QoS:

DDS_LivelinessQosPolicy (p. 1180)

Enumerator

DDS_AUTOMATIC_LIVELINESS_QOS	**[default]** The infrastructure will automatically signal liveliness for the **DDSDataWriter** (p. 1589) (s) at least as often as required by the **DDSDataWriter** (p. 1589) (S) `lease_duration`. A **DDSDataWriter** (p. 1589) with this setting does not need to take any specific action in order to be considered 'alive.' The **DDSDataWriter** (p. 1589) is only 'not alive' when the participant to which it belongs terminates (gracefully or not), or when there is a network problem that prevents the current participant from contacting that remote participant.
DDS_MANUAL_BY_PARTICIPANT_LIVELINESS_QOS	RTI Connext will assume that as long as at least one **DDSDataWriter** (p. 1589) belonging to the **DDSDomainParticipant** (p. 1619) (or the **DDSDomainParticipant** (p. 1619) itself) has asserted its liveliness, then the other DataWriters belonging to that same **DDSDomainParticipant** (p. 1619) are also alive. The user application takes responsibility to signal liveliness to RTI Connext either by calling **DDSDomainParticipant::assert_liveliness** (p. 1667), or by calling **DDSDataWriter::assert_liveliness** (p. 1597), or **FooDataWriter::write** (p. 1951) on any **DDSDataWriter** (p. 1589) belonging to the **DDSDomainParticipant** (p. 1619).
DDS_MANUAL_BY_TOPIC_LIVELINESS_QOS	RTI Connext will only assume liveliness of the **DDSDataWriter** (p. 1589) if the application has asserted liveliness of that **DDSDataWriter** (p. 1589) itself. The user application takes responsibility to signal liveliness to RTI Connext using the **DDSDataWriter::assert_liveliness** (p. 1597) method, or by writing some data.

7.166.3 Variable Documentation

7.166.3.1 DDS_LIVELINESS_QOS_POLICY_NAME

```
const char* const DDS_LIVELINESS_QOS_POLICY_NAME  [extern]
```

Stringified human-readable name for **DDS_LivelinessQosPolicy** (p. 1180).

7.167 LOCATORFILTER

<<***extension***>> *(p. 213)* The QoS policy used to report the configuration of a MultiChannel DataWriter as part of **DDS_PublicationBuiltinTopicData** (p. 1264).

Classes

- struct **DDS_LocatorFilter_t**

  *<<**extension**>> (p. 213) Specifies the configuration of an individual channel within a MultiChannel DataWriter.*
- struct **DDS_LocatorFilterSeq**

  *Declares IDL `sequence<` **DDS_LocatorFilter_t** (p. 1185) `>`.*
- struct **DDS_LocatorFilterQosPolicy**

  *<<**extension**>> (p. 213) The QoS policy used to report the configuration of a MultiChannel DataWriter as part of **DDS_PublicationBuiltinTopicData** (p. 1264).*

Variables

- const char ∗const **DDS_LOCATORFILTER_QOS_POLICY_NAME**

  *Stringified human-readable name for **DDS_LocatorFilterQosPolicy** (p. 1186).*

7.167.1 Detailed Description

*<<**extension**>> (p. 213)* The QoS policy used to report the configuration of a MultiChannel DataWriter as part of **DDS_PublicationBuiltinTopicData** (p. 1264).

7.167.2 Variable Documentation

7.167.2.1 DDS_LOCATORFILTER_QOS_POLICY_NAME

```
const char* const DDS_LOCATORFILTER_QOS_POLICY_NAME  [extern]
```

Stringified human-readable name for **DDS_LocatorFilterQosPolicy** (p. 1186).

7.168 LOGGING

*<<**extension**>> (p. 213)* Configures the RTI Connext logging facility.

Classes

- struct **DDS_LoggingQosPolicy**

  *<<**extension**>> (p. 213) Configures the RTI Connext logging facility.*

7.168.1 Detailed Description

<<***extension***>> *(p. 213)* Configures the RTI Connext logging facility.

7.169 MONITORING

<<***extension***>> *(p. 213)* Configures the use of RTI Monitoring Library 2.0 to collect and distribute RTI Connext telemetry data.

Classes

- struct **DDS_MonitoringDedicatedParticipantSettings**

  <<***extension***>> *(p. 213) Configures the use of a dedicated* ***DDSDomainParticipant*** *(p. 1619) to distribute the RTI Connext application telemetry data.*
- struct **DDS_MonitoringEventDistributionSettings**

  <<***extension***>> *(p. 213) Configures the distribution of event metrics.*
- struct **DDS_MonitoringPeriodicDistributionSettings**

  <<***extension***>> *(p. 213) Configures the distribution of periodic metrics.*
- struct **DDS_MonitoringLoggingDistributionSettings**

  <<***extension***>> *(p. 213) Configures the distribution of log messages.*
- struct **DDS_MonitoringDistributionSettings**

  <<***extension***>> *(p. 213) Configures the distribution of telemetry data.*
- struct **DDS_MonitoringMetricSelection**

  <<***extension***>> *(p. 213) This data structure is used to configure event and periodic metrics collection and distribution for a specific set of observable resources.*
- struct **DDS_MonitoringMetricSelectionSeq**

  Declares IDL `sequence` *<* ***DDS_MonitoringMetricSelection*** *(p. 1206) >*
- struct **DDS_MonitoringLoggingForwardingSettings**

  <<***extension***>> *(p. 213) Configures the forwarding levels of log messages for the different* ***NDDS_Config_LogFacility*** *(p. 727).*
- struct **DDS_MonitoringTelemetryData**

  <<***extension***>> *(p. 213) Configures the telemetry data that will be distributed.*
- struct **DDS_MonitoringQosPolicy**

  <<***extension***>> *(p. 213) Configures the use of RTI Monitoring Library 2.0 to collect and distribute RTI Connext telemetry data.*

Variables

- const char ∗const **DDS_MONITORING_QOS_POLICY_NAME**

  Stringified human-readable name for ***DDS_MonitoringQosPolicy*** *(p. 1210).*

7.169.1 Detailed Description

<<***extension***>> *(p. 213)* Configures the use of RTI Monitoring Library 2.0 to collect and distribute RTI Connext telemetry data.

7.169.2 Variable Documentation

7.169.2.1 DDS_MONITORING_QOS_POLICY_NAME

```
const char* const DDS_MONITORING_QOS_POLICY_NAME  [extern]
```

Stringified human-readable name for **DDS_MonitoringQosPolicy** (p. 1210).

7.170 MULTICHANNEL

*<<**extension**>> (p. 213)* Configures the ability of a DataWriter to send data on different multicast groups (addresses) based on the value of the data.

Classes

- struct **DDS_ChannelSettings_t**

  *<<**extension**>> (p. 213) Type used to configure the properties of a channel.*
- struct **DDS_ChannelSettingsSeq**

  *Declares IDL `sequence`< **DDS_ChannelSettings_t** (p. 822) >*
- struct **DDS_MultiChannelQosPolicy**

  *<<**extension**>> (p. 213) Configures the ability of a DataWriter to send data on different multicast groups (addresses) based on the value of the data.*

Variables

- const char *const **DDS_MULTICHANNEL_QOS_POLICY_NAME**

  *Stringified human-readable name for **DDS_MultiChannelQosPolicy** (p. 1214).*

7.170.1 Detailed Description

*<<**extension**>> (p. 213)* Configures the ability of a DataWriter to send data on different multicast groups (addresses) based on the value of the data.

7.170.2 Variable Documentation

7.170.2.1 DDS_MULTICHANNEL_QOS_POLICY_NAME

```
const char* const DDS_MULTICHANNEL_QOS_POLICY_NAME  [extern]
```

Stringified human-readable name for **DDS_MultiChannelQosPolicy** (p. 1214).

7.171 OWNERSHIP

<<***spec***>> *(p. 209)* Specifies whether it is allowed for multiple **DDSDataWriter** (p. 1589) (s) to write the same instance of the data and if so, how these modifications should be arbitrated.

Classes

- struct **DDS_OwnershipQosPolicy**

  <<***spec***>> *(p. 209) Specifies whether it is allowed for multiple **DDSDataWriter** (p. 1589) (s) to write the same instance of the data and if so, how these modifications should be arbitrated.*

Enumerations

- enum **DDS_OwnershipQosPolicyKind** {
  DDS_SHARED_OWNERSHIP_QOS ,
  DDS_EXCLUSIVE_OWNERSHIP_QOS }

  Kinds of ownership.

Variables

- const char ∗const **DDS_OWNERSHIP_QOS_POLICY_NAME**

  *Stringified human-readable name for **DDS_OwnershipQosPolicy** (p. 1222).*

7.171.1 Detailed Description

<<***spec***>> *(p. 209)* Specifies whether it is allowed for multiple **DDSDataWriter** (p. 1589) (s) to write the same instance of the data and if so, how these modifications should be arbitrated.

7.171.2 Enumeration Type Documentation

7.171.2.1 DDS_OwnershipQosPolicyKind

```
enum DDS_OwnershipQosPolicyKind
```

Kinds of ownership.

QoS:

DDS_OwnershipQosPolicy (p. 1222)

Enumerator

DDS_SHARED_OWNERSHIP_QOS	**[default]** Indicates shared ownership for each instance. Multiple writers are allowed to update the same instance and all the updates are made available to the readers. In other words there is no concept of an owner for the instances. This is the **default** behavior if the **OWNERSHIP** (p. 612) policy is not specified or supported.
DDS_EXCLUSIVE_OWNERSHIP_QOS	Indicates each instance can only be owned by one **DDSDataWriter** (p. 1589), but the owner of an instance can change dynamically. The selection of the owner is controlled by the setting of the **OWNERSHIP_STRENGTH** (p. 613) policy. The owner is always set to be the highest-strength **DDSDataWriter** (p. 1589) object among the ones currently active (as determined by the **LIVELINESS** (p. 607)).

7.171.3 Variable Documentation

7.171.3.1 DDS_OWNERSHIP_QOS_POLICY_NAME

```
const char* const DDS_OWNERSHIP_QOS_POLICY_NAME  [extern]
```

Stringified human-readable name for **DDS_OwnershipQosPolicy** (p. 1222).

7.172 OWNERSHIP_STRENGTH

<<***spec***>> *(p. 209)* Specifies the value of the strength used to arbitrate among multiple **DDSDataWriter** (p. 1589) objects that attempt to modify the same instance of a data type (identified by **DDSTopic** (p. 1885) + key).

Classes

- struct **DDS_OwnershipStrengthQosPolicy**

  *Specifies the value of the strength used to arbitrate among multiple **DDSDataWriter** (p. 1589) objects that attempt to modify the same instance of a data type (identified by **DDSTopic** (p. 1885) + key).*

Variables

- const char *const **DDS_OWNERSHIPSTRENGTH_QOS_POLICY_NAME**

  *Stringified human-readable name for **DDS_OwnershipStrengthQosPolicy** (p. 1227).*

7.172.1 Detailed Description

<<***spec***>> *(*p. *209)* Specifies the value of the strength used to arbitrate among multiple **DDSDataWriter** (p. 1589) objects that attempt to modify the same instance of a data type (identified by **DDSTopic** (p. 1885) + key).

7.172.2 Variable Documentation

7.172.2.1 DDS_OWNERSHIPSTRENGTH_QOS_POLICY_NAME

```
const char* const DDS_OWNERSHIPSTRENGTH_QOS_POLICY_NAME  [extern]
```

Stringified human-readable name for **DDS_OwnershipStrengthQosPolicy** (p. 1227).

7.173 PARTITION

<<***spec***>> *(*p. *209)* Set of strings that introduces logical partitions in **DDSDomainParticipant** (p. 1619), **DDSPublisher** (p. 1818), or **DDSSubscriber** (p. 1860) entities.

Classes

- struct **DDS_PartitionQosPolicy**

  <<***spec***>> *(*p. *209) Set of strings that introduces logical partitions in **DDSDomainParticipant** (*p. *1619), **DDSPublisher** (*p. *1818), or **DDSSubscriber** (*p. *1860) entities.*

Variables

- const char ∗const **DDS_PARTITION_QOS_POLICY_NAME**

  *Stringified human-readable name for **DDS_PartitionQosPolicy** (*p. *1240).*

7.173.1 Detailed Description

<<***spec***>> *(*p. *209)* Set of strings that introduces logical partitions in **DDSDomainParticipant** (p. 1619), **DDSPublisher** (p. 1818), or **DDSSubscriber** (p. 1860) entities.

7.173.2 Variable Documentation

7.173.2.1 DDS_PARTITION_QOS_POLICY_NAME

```
const char* const DDS_PARTITION_QOS_POLICY_NAME  [extern]
```

Stringified human-readable name for **DDS_PartitionQosPolicy** (p. 1240).

7.174 PRESENTATION

<<***spec***>> *(p. 209)* Specifies how the samples representing changes to data instances are presented to a subscribing application.

Classes

- struct **DDS_PresentationQosPolicy**

  <<***spec***>> *(p. 209) Specifies how the samples representing changes to data instances are presented to a subscribing application.*

Enumerations

- enum **DDS_PresentationQosPolicyAccessScopeKind** {
  DDS_INSTANCE_PRESENTATION_QOS ,
  DDS_TOPIC_PRESENTATION_QOS ,
  DDS_GROUP_PRESENTATION_QOS ,
  DDS_HIGHEST_OFFERED_PRESENTATION_QOS }

  Kinds of presentation "access scope".

Variables

- const char ∗const **DDS_PRESENTATION_QOS_POLICY_NAME**

  *Stringified human-readable name for **DDS_PresentationQosPolicy** (p. 1249).*

7.174.1 Detailed Description

<<***spec***>> *(p. 209)* Specifies how the samples representing changes to data instances are presented to a subscribing application.

7.174.2 Enumeration Type Documentation

7.174.2.1 DDS_PresentationQosPolicyAccessScopeKind

```
enum DDS_PresentationQosPolicyAccessScopeKind
```

Kinds of presentation "access scope".

Access scope determines the largest scope spanning the entities for which the order and coherency of changes can be preserved.

QoS:

DDS_PresentationQosPolicy (p. 1249)

Enumerator

DDS_INSTANCE_PRESENTATION_QOS	**[default]** Scope spans only a single instance. Indicates that changes to one instance need not be coherent nor ordered with respect to changes to any other instance. In other words, order and coherent changes apply to each instance separately.
DDS_TOPIC_PRESENTATION_QOS	Scope spans all instances within the same **DDSDataWriter** (p. 1589), but not across instances in different **DDSDataWriter** (p. 1589) entities.
DDS_GROUP_PRESENTATION_QOS	Scope spans all instances belonging to **DDSDataWriter** (p. 1589) entities within the same **DDSPublisher** (p. 1818).
DDS_HIGHEST_OFFERED_PRESENTATION_QOS	This value only applies to a **DDSSubscriber** (p. 1860). The **DDSSubscriber** (p. 1860) will use the access scope specified by each remote **DDSPublisher** (p. 1818).

7.174.3 Variable Documentation

7.174.3.1 DDS_PRESENTATION_QOS_POLICY_NAME

```
const char* const DDS_PRESENTATION_QOS_POLICY_NAME  [extern]
```

Stringified human-readable name for **DDS_PresentationQosPolicy** (p. 1249).

7.175 PROFILE

<<***extension***>> *(p. 213)* Configures the way that XML documents containing QoS profiles are loaded by RTI Connext.

Classes

- struct **DDS_ProfileQosPolicy**

  <<***extension***>> *(p. 213) Configures the way that XML documents containing QoS profiles are loaded by RTI Connext.*

Variables

- const char *const **DDS_PROFILE_QOS_POLICY_NAME**

  *Stringified human-readable name for **DDS_ProfileQosPolicy** (p. 1257).*

7.175.1 Detailed Description

<<***extension***>> *(p. 213)* Configures the way that XML documents containing QoS profiles are loaded by RTI Connext.

7.175.2 Variable Documentation

7.175.2.1 DDS_PROFILE_QOS_POLICY_NAME

```
const char* const DDS_PROFILE_QOS_POLICY_NAME  [extern]
```

Stringified human-readable name for **DDS_ProfileQosPolicy** (p. 1257).

7.176 PROPERTY

<<***extension***>> *(p. 213)* Stores (name, value) pairs that can be used to configure certain parameters of RTI Connext that are not exposed through formal QoS policies. Can also be used to store and propagate application-specific name/value pairs that can be retrieved by user code during discovery.

Classes

- struct **DDS_Property_t**

  Properties are name/value pairs objects.
- struct **DDS_PropertySeq**

  Declares IDL `sequence` *<* ***DDS_Property_t*** *(p. 1259) >*
- struct **DDS_PropertyQosPolicy**

  *<<**extension**>> (p. 213) Stores name/value(string) pairs that can be used to configure certain parameters of RTI Connext that are not exposed through formal QoS policies. Can also be used to store and propagate application-specific name/value pairs that can be retrieved by user code during discovery.*
- class **DDSPropertyQosPolicyHelper**

  Policy helpers that facilitate management of the properties in the input policy.

Typedefs

- typedef struct **DDS_PropertyQosPolicy** **DDS_PropertyQosPolicy**

  *<<**extension**>> (p. 213) Stores name/value(string) pairs that can be used to configure certain parameters of RTI Connext that are not exposed through formal QoS policies. Can also be used to store and propagate application-specific name/value pairs that can be retrieved by user code during discovery.*
- typedef enum **DDS_PropertyQosPolicyMutability** **DDS_PropertyQosPolicyMutability**

  Determines if, and when, the value of a property can change.

Enumerations

- enum **DDS_PropertyQosPolicyMutability** {
  DDS_PROPERTY_QOS_MUTABLE ,
  DDS_PROPERTY_QOS_MUTABLE_UNTIL_ENABLE ,
  DDS_PROPERTY_QOS_IMMUTABLE }

  Determines if, and when, the value of a property can change.

Functions

- **DDS_PropertyQosPolicyMutability** **DDS_PropertyQosPolicyHelper_get_property_mutability** (const char ∗name, const struct **DDS_PropertyQosPolicy** ∗policy)

  Returns the mutability type of a property.
- static **DDS_Long** **DDSPropertyQosPolicyHelper::get_number_of_properties** (**DDS_PropertyQosPolicy** &policy)

  Gets the number of properties in the input policy.
- static **DDS_ReturnCode_t** **DDSPropertyQosPolicyHelper::assert_property** (**DDS_PropertyQosPolicy** &policy, const char ∗name, const char ∗value, **DDS_Boolean** propagate)

  Asserts the property identified by name in the input policy.
- static **DDS_ReturnCode_t** **DDSPropertyQosPolicyHelper::add_property** (**DDS_PropertyQosPolicy** &policy, const char ∗name, const char ∗value, **DDS_Boolean** propagate)

  Adds a new property to the input policy.
- static **DDS_ReturnCode_t** **DDSPropertyQosPolicyHelper::assert_pointer_property** (**DDS_PropertyQos↩ Policy** &policy, const char ∗name, const void ∗pointer)

  Asserts the property identified by name in the input policy. Used when the property to store is a pointer.
- static **DDS_ReturnCode_t** **DDSPropertyQosPolicyHelper::add_pointer_property** (**DDS_PropertyQos↩ Policy** &policy, const char ∗name, const void ∗pointer)

  Adds a new property to the input policy. Used when the property to store is a pointer.
- static struct **DDS_Property_t** ∗ **DDSPropertyQosPolicyHelper::lookup_property** (**DDS_PropertyQosPolicy** &policy, const char ∗name)

  Searches for a property in the input policy given its name.
- static **DDS_ReturnCode_t** **DDSPropertyQosPolicyHelper::remove_property** (**DDS_PropertyQosPolicy** &policy, const char ∗name)

  Removes a property from the input policy.
- static **DDS_ReturnCode_t** **DDSPropertyQosPolicyHelper::get_properties** (**DDS_PropertyQosPolicy** &policy, struct **DDS_PropertySeq** &properties, const char ∗name_prefix)

  Retrieves a list of properties whose names match the input prefix.

Variables

- const char ∗const **DDS_PROPERTY_QOS_POLICY_NAME**

  *Stringified human-readable name for **DDS_PropertyQosPolicy** (p. 1260).*

7.176.1 Detailed Description

<<***extension***>> *(p. 213)* Stores (name, value) pairs that can be used to configure certain parameters of RTI Connext that are not exposed through formal QoS policies. Can also be used to store and propagate application-specific name/value pairs that can be retrieved by user code during discovery.

RTI Connext will automatically set some system properties in the **DDS_PropertyQosPolicy** (p. 1260) associated with a **DDS_DomainParticipantQos** (p. 971). See **System Properties** (p. 389) for additional details.

7.176.2 Typedef Documentation

7.176.2.1 DDS_PropertyQosPolicy

`typedef struct` **`DDS_PropertyQosPolicy DDS_PropertyQosPolicy`**

<<***extension***>> *(p. 213)* Stores name/value(string) pairs that can be used to configure certain parameters of RTI Connext that are not exposed through formal QoS policies. Can also be used to store and propagate application-specific name/value pairs that can be retrieved by user code during discovery.

Entity:

DDSDomainParticipant (p. 1619) **DDSDataReader** (p. 1556) **DDSDataWriter** (p. 1589)

Properties:

RxO (p. 556) = N/A;
Changeable (p. 556) = **YES** (p. 556)

See also

DDSDomainParticipant::get_builtin_subscriber (p. 1661)

7.176.3 Usage

The PROPERTY QoS policy can be used to associate a set of properties in the form of (name, value) pairs with a **DDSDataReader** (p. 1556), **DDSDataWriter** (p. 1589), or **DDSDomainParticipant** (p. 1619). This is similar to the **DDS_UserDataQosPolicy** (p. 1504), except this policy uses (name, value) pairs, and you can select whether or not a particular pair should be propagated (included in the builtin topic).

You can find a complete list of predefined properties in the `Property Reference Guide`.

This QoS policy may be used to configure:

- Durable Writer History, see **Configuring Durable Writer History** (p. 198)
- Durable Reader State, see **Configuring Durable Reader State** (p. 200)
- Builtin Transport Plugins, see **UDPv4 Transport Property Names in Property QoS Policy of Domain Participant** (p. 465), **UDPv6 Transport Property Names in Property QoS Policy of Domain Participant** (p. 481), and **Shared Memory Transport Property Names in Property QoS Policy of Domain Participant** (p. 459)
- Extension Transport Plugins, see **Loading Transport Plugins through Property QoS Policy of Domain Participant** (p. 178)
- **Clock Selection** (p. 40)

In addition, you may add your own name/value pairs to the Property QoS policy of an Entity. Via this QoS policy, you can direct RTI Connext to propagate these name/value pairs with the discovery information for the Entity. Applications that discover the Entity can then access the user-specific name/value pairs in the discovery information of the remote Entity. This allows you to add meta-information about an Entity for application-specific use, for example, authentication/authorization certificates (which can also be done using the **DDS_UserDataQosPolicy** (p. 1504) or **DDS_Group↩ DataQosPolicy** (p. 1160)).

7.176.3.1 Reasons for Using the PropertyQosPolicy

- Supports dynamic loading of extension transports
- Supports multiple instances of the builtin transports
- Allows full pluggable transport configuration for non-C/C++ language bindings (Java, .NET, etc.)
- Avoids the process of creating entities disabled, changing their QoS settings, then enabling them
- Allows selection of clock

Some of the RTI Connext capabilities configurable via the Property QoS policy can also be configured in code via APIs. However, the Property QoS policy allows you to configure those parameters via XML files. In addition, some of the configuration APIs will only work if the Entity was created in a disabled state and then enabled after the configuration change was applied. By configuring those parameters using the Property QoS policy during entity creation, you avoid the additional work of first creating a disabled entity and then enabling it afterwards.

There are helper functions to facilitate working with properties, see the **PROPERTY** (p. 617) page.

7.176.3.2 DDS_PropertyQosPolicyMutability

`typedef enum` **`DDS_PropertyQosPolicyMutability`** **`DDS_PropertyQosPolicyMutability`**

Determines if, and when, the value of a property can change.

7.176.4 Enumeration Type Documentation

7.176.4.1 DDS_PropertyQosPolicyMutability

`enum` **`DDS_PropertyQosPolicyMutability`**

Determines if, and when, the value of a property can change.

Enumerator

DDS_PROPERTY_QOS_MUTABLE	The property is mutable: it can be changed at any time.
DDS_PROPERTY_QOS_MUTABLE_UNTIL_ENABLE	The property can only be changed before enabling the entity.
DDS_PROPERTY_QOS_IMMUTABLE	The property is immutable: it can only be specified when creating the entity.

7.176.5 Function Documentation

7.176.5.1 DDS_PropertyQosPolicyHelper_get_property_mutability()

```
DDS_PropertyQosPolicyMutability DDS_PropertyQosPolicyHelper_get_property_mutability (
            const char * name,
            const struct  DDS_PropertyQosPolicy * policy )
```

Returns the mutability type of a property.

If the property does not exist, **DDS_PROPERTY_QOS_MUTABLE** (p. 620) is returned. Otherwise, it returns the mutability type of the specified property. See **DDS_PropertyQosPolicyMutability** (p. 620) for more information of the different mutability types.

Parameters

name	*<<**in**>> (p. 433)*. The name of the property for which we want to know the mutability type.
policy	*<<**in**>> (p. 433)*. These are the properties in which we are going to look for the property with the provided name.

Returns

The mutability type of the input property.

7.176.5.2 get_number_of_properties()

```
static  DDS_Long DDSPropertyQosPolicyHelper::get_number_of_properties (
              DDS_PropertyQosPolicy & policy )  [static]
```

Gets the number of properties in the input policy.

Precondition

policy cannot be NULL.

Parameters

policy	*<<**in**>> (p. 433)* Input policy.

Returns

Number of properties.

7.176.5.3 assert_property()

```
static DDS_ReturnCode_t DDSPropertyQosPolicyHelper::assert_property (
            DDS_PropertyQosPolicy & policy,
            const char * name,
            const char * value,
            DDS_Boolean propagate )  [static]
```

Asserts the property identified by name in the input policy.

If the property already exists, this function replaces its current value with the new one.

If the property identified by name does not exist, this function adds it to the property set.

This function increases the maximum number of elements of the policy sequence when this number is not enough to store the new property.

Precondition

policy, name and value cannot be NULL.

Parameters

policy	<<***in***>> *(p. 433)* Input policy.
name	<<***in***>> *(p. 433)* Property name.
value	<<***in***>> *(p. 433)* Property value.
propagate	<<***in***>> *(p. 433)* Indicates if the property will be propagated on discovery.

Exceptions

One	of the **Standard Return Codes** (p. 532), **DDS_RETCODE_OUT_OF_RESOURCES** (p. 533)

7.176.5.4 add_property()

```
static DDS_ReturnCode_t DDSPropertyQosPolicyHelper::add_property (
            DDS_PropertyQosPolicy & policy,
            const char * name,
```

```
            const char * value,
             DDS_Boolean propagate )  [static]
```

Adds a new property to the input policy.

This function will allocate memory to store the (name, value) pair. The memory allocated is owned by RTI Connext.

If the maximum number of elements of the policy sequence is not enough to store the new property, this function will increase it.

If the property already exists the function fails with **DDS_RETCODE_PRECONDITION_NOT_MET** (p. 532).

Precondition

policy, name and value cannot be NULL.

The property is not in the policy.

Parameters

policy	<<***in***>> *(p. 433)* Input policy.
name	<<***in***>> *(p. 433)* Property name.
value	<<***in***>> *(p. 433)* Property value.
propagate	<<***in***>> *(p. 433)* Indicates if the property will be propagated on discovery.

Exceptions

One	of the **Standard Return Codes** (p. 532), **DDS_RETCODE_PRECONDITION_NOT_MET** (p. 532), **DDS_RETCODE_OUT_OF_RESOURCES** (p. 533)

7.176.5.5 assert_pointer_property()

```
static  DDS_ReturnCode_t DDSPropertyQosPolicyHelper::assert_pointer_property (
             DDS_PropertyQosPolicy & policy,
            const char * name,
            const void * pointer )  [static]
```

Asserts the property identified by name in the input policy. Used when the property to store is a pointer.

This is a function similar to **DDSPropertyQosPolicyHelper::assert_property** (p. 622). However, instead of passing a stringified version of the pointer, this function receives a pointer as the value.

If the property already exists, this function replaces its current value with the new one.

If the property identified by name does not exist, this function adds it to the property set.

This function increases the maximum number of elements of the policy sequence when this number is not enough to store the new property.

The properties asserted by this function will not be propagated on discovery.

Precondition

policy and name cannot be NULL.

Parameters

policy	<<***in***>> *(*p. *433)* Input policy.
name	<<***in***>> *(*p. *433)* Property name.
pointer	<<***in***>> *(*p. *433)* The pointer to store in the property.

Returns

One of the **Standard Return Codes** (p. 532) or **DDS_RETCODE_OUT_OF_RESOURCES** (p. 533).

7.176.5.6 add_pointer_property()

```
static DDS_ReturnCode_t DDSPropertyQosPolicyHelper::add_pointer_property (
            DDS_PropertyQosPolicy & policy,
            const char * name,
            const void * pointer )  [static]
```

Adds a new property to the input policy. Used when the property to store is a pointer.

This is a function similar to **DDSPropertyQosPolicyHelper::add_property** (p. 622). However, instead of passing a stringified version of the pointer, this function receives a pointer as the value.

This function will allocate memory to store the (name, value) pair. The memory allocated is owned by RTI Connext.

If the maximum number of elements of the policy sequence is not enough to store the new property, this function will increase it.

If the property already exists the function fails with **DDS_RETCODE_PRECONDITION_NOT_MET** (p. 532).

The properties added by this function will not be propagated on discovery.

Precondition

policy and name cannot be NULL.

The property is not in the policy.

Parameters

policy	<<***in***>> *(*p. *433)* Input policy.
name	<<***in***>> *(*p. *433)* Property name.
pointer	<<***in***>> *(*p. *433)* The pointer to store in the property.

Returns

One of the **Standard Return Codes** (p. 532) or **DDS_RETCODE_OUT_OF_RESOURCES** (p. 533) or **DDS_↩ RETCODE_PRECONDITION_NOT_MET** (p. 532)

7.176.5.7 lookup_property()

```
static struct  DDS_Property_t * DDSPropertyQosPolicyHelper::lookup_property (
             DDS_PropertyQosPolicy & policy,
            const char * name )  [static]
```

Searches for a property in the input policy given its name.

Precondition

policy, name and value cannot be NULL.

Parameters

policy	<<***in***>> *(p. 433)* Input policy.
name	<<***in***>> *(p. 433)* Property name.

Returns

The function returns the first property with the given name. If such a property does not exist, the function returns NULL.

7.176.5.8 remove_property()

```
static  DDS_ReturnCode_t DDSPropertyQosPolicyHelper::remove_property (
             DDS_PropertyQosPolicy & policy,
            const char * name )  [static]
```

Removes a property from the input policy.

If the property does not exist, the function fails with **DDS_RETCODE_PRECONDITION_NOT_MET** (p. 532).

Precondition

policy and name cannot be NULL.

The property is in the policy.

Parameters

policy	*<<**in**>> (*p. *433)* Input policy.
name	*<<**in**>> (*p. *433)* Property name.

Exceptions

One	of the **Standard Return Codes** (p. 532), **DDS_RETCODE_PRECONDITION_NOT_MET** (p. 532)

7.176.5.9 get_properties()

```
static DDS_ReturnCode_t DDSPropertyQosPolicyHelper::get_properties (
            DDS_PropertyQosPolicy & policy,
            struct DDS_PropertySeq & properties,
            const char * name_prefix ) [static]
```

Retrieves a list of properties whose names match the input prefix.

If the properties sequence doesn't own its buffer, and its maximum is less than the total number of properties matching the input prefix, it will be filled up to its maximum and fail with an error of **DDS_RETCODE_OUT_OF_RESOURCES** (p. 533).

Precondition

policy, properties and name_prefix cannot be NULL.

Parameters

policy	*<<**in**>> (*p. *433)* Input policy.
properties	*<<**inout**>> (*p. *433)* A **DDS_PropertySeq** (p. 1262) object where the set or list of properties will be returned.
name_prefix	Name prefix.

Exceptions

One	of the **Standard Return Codes** (p. 532), **DDS_RETCODE_PRECONDITION_NOT_MET** (p. 532)

7.176.6 Variable Documentation

7.176.6.1 DDS_PROPERTY_QOS_POLICY_NAME

```
const char* const DDS_PROPERTY_QOS_POLICY_NAME  [extern]
```

Stringified human-readable name for **DDS_PropertyQosPolicy** (p. 1260).

7.177 PUBLISH_MODE

<<***extension***>> *(p. 213)* Specifies how RTI Connext sends application data on the network. This QoS policy can be used to tell RTI Connext to use its *own* thread to send data, instead of the user thread.

Classes

- struct **DDS_PublishModeQosPolicy**

  <<***extension***>> *(p. 213) Specifies how RTI Connext sends application data on the network. This QoS policy can be used to tell RTI Connext to use its own thread to send data, instead of the user thread.*

Macros

- #define **DDS_PUBLICATION_PRIORITY_UNDEFINED**

  *Initializer value for **DDS_PublishModeQosPolicy::priority** (p. 1282) and/or **DDS_ChannelSettings_t::priority** (p. 823).*
- #define **DDS_PUBLICATION_PRIORITY_AUTOMATIC**

  *Constant value for **DDS_PublishModeQosPolicy::priority** (p. 1282) and/or **DDS_ChannelSettings_t::priority** (p. 823).*

Enumerations

- enum **DDS_PublishModeQosPolicyKind** {
  DDS_SYNCHRONOUS_PUBLISH_MODE_QOS ,
  DDS_ASYNCHRONOUS_PUBLISH_MODE_QOS }

  Kinds of publishing mode.

Variables

- const char ∗const **DDS_PUBLISHMODE_QOS_POLICY_NAME**

  *Stringified human-readable name for **DDS_PublishModeQosPolicy** (p. 1279).*

7.177.1 Detailed Description

<<***extension***>> *(p. 213)* Specifies how RTI Connext sends application data on the network. This QoS policy can be used to tell RTI Connext to use its *own* thread to send data, instead of the user thread.

7.177.2 Macro Definition Documentation

7.177.2.1 DDS_PUBLICATION_PRIORITY_UNDEFINED

```
#define DDS_PUBLICATION_PRIORITY_UNDEFINED
```

Initializer value for **DDS_PublishModeQosPolicy::priority** (p. 1282) and/or **DDS_ChannelSettings_t::priority** (p. 823).

When assigned this value, the publication priority of the data writer, or channel of a multi-channel data writer, will be set to the lowest possible value. For muilti-channel data writers, if either the data writer or channel priority is NOT set to this value, then the publication priority of the entity will be set to the defined value.

7.177.2.2 DDS_PUBLICATION_PRIORITY_AUTOMATIC

```
#define DDS_PUBLICATION_PRIORITY_AUTOMATIC
```

Constant value for **DDS_PublishModeQosPolicy::priority** (p. 1282) and/or **DDS_ChannelSettings_t::priority** (p. 823).

When assigned this value, the publication priority of the data writer, or channel of a multi-channel data writer, will be set to the largest priority value of any sample currently queued for publication by the data writer or data writer channel.

7.177.3 Enumeration Type Documentation

7.177.3.1 DDS_PublishModeQosPolicyKind

```
enum DDS_PublishModeQosPolicyKind
```

Kinds of publishing mode.

QoS:

DDS_PublishModeQosPolicy (p. 1279)

Enumerator

DDS_SYNCHRONOUS_PUBLISH_MODE_QOS	Indicates to send data synchronously. If **DDS_DataWriterProtocolQosPolicy::push_on_write** (p. 894) is **DDS_BOOLEAN_TRUE** (p. 513), data is sent immediately in the context of **FooDataWriter::write** (p. 1951). As data is sent immediately in the context of the user thread, no flow control is applied. **See also** **DDS_DataWriterProtocolQosPolicy::push_on_write** (p. 894) **[default]** for **DDSDataWriter** (p. 1589)
DDS_ASYNCHRONOUS_PUBLISH_MODE_QOS	Indicates to send data asynchronously. Configures the **DDSDataWriter** (p. 1589) to delegate the task of data transmission to a separate publishing thread. The **FooDataWriter::write** (p. 1951) call does not send the data, but instead schedules the data to be sent later by its associated **DDSPublisher** (p. 1818). Each **DDSPublisher** (p. 1818) uses its dedicated publishing thread (**DDS_PublisherQos::asynchronous_publisher** (p. 1279)) to send data for all its asynchronous DataWriters. For each asynchronous DataWriter, the associated **DDSFlowController** (p. 1736) determines when the publishing thread is allowed to send the data. **DDSDataWriter::wait_for_asynchronous_publishing** (p. 1603) and **DDSPublisher::wait_for_asynchronous_publishing** (p. 1835) enable you to determine when the data has actually been sent. **See also** **DDSFlowController** (p. 1736) **DDS_HistoryQosPolicy** (p. 1162) **DDSDataWriter::wait_for_asynchronous_publishing** (p. 1603) **DDSPublisher::wait_for_asynchronous_publishing** (p. 1835) **NDDS_Transport_Property_t::gather_send_buffer_count_max** (p. 2047)

7.177.4 Variable Documentation

7.177.4.1 DDS_PUBLISHMODE_QOS_POLICY_NAME

```
const char* const DDS_PUBLISHMODE_QOS_POLICY_NAME  [extern]
```

Stringified human-readable name for **DDS_PublishModeQosPolicy** (p. 1279).

7.178 READER_DATA_LIFECYCLE

<<***spec***>> *(p. 209)* Controls how a DataReader manages the lifecycle of the data that it has received.

Classes

- struct **DDS_ReaderDataLifecycleQosPolicy**
  <<***spec***>> *(p. 209) Controls how a DataReader manages the lifecycle of the data that it has received.*

Variables

- const char *const **DDS_READERDATALIFECYCLE_QOS_POLICY_NAME**
  *Stringified human-readable name for **DDS_ReaderDataLifecycleQosPolicy** (p. 1289).*

7.178.1 Detailed Description

<<***spec***>> *(p. 209)* Controls how a DataReader manages the lifecycle of the data that it has received.

7.178.2 Variable Documentation

7.178.2.1 DDS_READERDATALIFECYCLE_QOS_POLICY_NAME

```
const char* const DDS_READERDATALIFECYCLE_QOS_POLICY_NAME  [extern]
```

Stringified human-readable name for **DDS_ReaderDataLifecycleQosPolicy** (p. 1289).

7.179 RECEIVER_POOL

<<***extension***>> *(p. 213)* Configures threads used by RTI Connext to receive and process data from transports (for example, UDP sockets).

Classes

- struct **DDS_ReceiverPoolQosPolicy**

  <<***extension***>> *(p. 213) Configures threads used by RTI Connext to receive and process data from transports (for example, UDP sockets).*

Variables

- const **DDS_Long DDS_LENGTH_AUTO**

  A special value indicating that the actual value will be automatically resolved.
- const char *const **DDS_RECEIVERPOOL_QOS_POLICY_NAME**

  *Stringified human-readable name for **DDS_ReceiverPoolQosPolicy** (p. 1293).*

7.179.1 Detailed Description

<<***extension***>> *(p. 213)* Configures threads used by RTI Connext to receive and process data from transports (for example, UDP sockets).

7.179.2 Variable Documentation

7.179.2.1 DDS_LENGTH_AUTO

```
const  DDS_Long DDS_LENGTH_AUTO  [extern]
```

A special value indicating that the actual value will be automatically resolved.

7.179.2.2 DDS_RECEIVERPOOL_QOS_POLICY_NAME

```
const char* const DDS_RECEIVERPOOL_QOS_POLICY_NAME  [extern]
```

Stringified human-readable name for **DDS_ReceiverPoolQosPolicy** (p. 1293).

7.180 RELIABILITY

<<***spec***>> *(p. 209)* Indicates the level of reliability offered/requested by RTI Connext.

Classes

- struct **DDS_ReliabilityQosPolicy**

  *<<**extension**>> (p. 213) <<**spec**>> (p. 209) Indicates the level of reliability offered/requested by RTI Connext.*

Enumerations

- enum **DDS_ReliabilityQosPolicyKind** {
  DDS_BEST_EFFORT_RELIABILITY_QOS ,
  DDS_RELIABLE_RELIABILITY_QOS }

  Kinds of reliability.
- enum **DDS_ReliabilityQosPolicyAcknowledgmentModeKind** {
  DDS_PROTOCOL_ACKNOWLEDGMENT_MODE ,
  DDS_APPLICATION_AUTO_ACKNOWLEDGMENT_MODE ,
  DDS_APPLICATION_ORDERED_ACKNOWLEDGMENT_MODE ,
  DDS_APPLICATION_EXPLICIT_ACKNOWLEDGMENT_MODE }

  *<<**extension**>> (p. 213) Kinds of acknowledgment.*
- enum **DDS_InstanceStateConsistencyKind** {
  DDS_NO_RECOVER_INSTANCE_STATE_CONSISTENCY ,
  DDS_RECOVER_INSTANCE_STATE_CONSISTENCY }

  *<<**extension**>> (p. 213) Whether instance state consistency is enabled.*

Variables

- const char *const **DDS_RELIABILITY_QOS_POLICY_NAME**

  *Stringified human-readable name for **DDS_ReliabilityQosPolicy** (p. 1295).*

7.180.1 Detailed Description

*<<**spec**>> (p. 209)* Indicates the level of reliability offered/requested by RTI Connext.

7.180.2 Enumeration Type Documentation

7.180.2.1 DDS_ReliabilityQosPolicyKind

`enum` **`DDS_ReliabilityQosPolicyKind`**

Kinds of reliability.

QoS:

DDS_ReliabilityQosPolicy (p. 1295)

Enumerator

DDS_BEST_EFFORT_RELIABILITY_QOS	Indicates that it is acceptable to not retry propagation of any samples. Presumably new values for the samples are generated often enough that it is not necessary to re-send or acknowledge any samples. **[default]** for **DDSDataReader** (p. 1556) and **DDSTopic** (p. 1885)
DDS_RELIABLE_RELIABILITY_QOS	Specifies RTI Connext will attempt to deliver all samples in its history. Missed samples may be retried. In steady-state (no modifications communicated via the **DDSDataWriter** (p. 1589)), RTI Connext guarantees that all samples in the **DDSDataWriter** (p. 1589) history will eventually be delivered to all the **DDSDataReader** (p. 1556) objects (subject to timeouts that indicate loss of communication with a particular **DDSSubscriber** (p. 1860)). Outside steady state, the **HISTORY** (p. 603) and **RESOURCE_LIMITS** (p. 635) policies will determine how samples become part of the history and whether samples can be discarded from it. **[default]** for **DDSDataWriter** (p. 1589)

7.180.2.2 DDS_ReliabilityQosPolicyAcknowledgmentModeKind

`enum` **`DDS_ReliabilityQosPolicyAcknowledgmentModeKind`**

<<***extension***>> *(p. 213)* Kinds of acknowledgment.

QoS:

DDS_ReliabilityQosPolicy (p. 1295)

Enumerator

DDS_PROTOCOL_ACKNOWLEDGMENT_MODE	Samples are acknowledged by RTPS protocol. Samples are acknowledged according to the Real-Time Publish-Subscribe (RTPS) interoperability protocol.
DDS_APPLICATION_AUTO_ACKNOWLEDGMENT_↩ MODE	Samples are acknowledged automatically after a subscribing application has accessed them. A sample received by a **FooDataReader** (p. 1917) is acknowledged after the subscribing application accesses it, either through calling **FooDataReader::take** (p. 1920) or **FooDataReader::read** (p. 1919) on the DDS sample. If the read or take operation loans the samples, the acknowledgment is done after **FooDataReader::return_loan** (p. 1940) is called. Otherwise, for read or take operations that make a copy, acknowledgment is done after the read or take operations are executed.

Enumerator

DDS_APPLICATION_EXPLICIT_↩ ACKNOWLEDGMENT_MODE	Samples are acknowledged after the subscribing application explicitly calls acknowledge on the samples. Samples received by a **DDSDataReader** (p. 1556) are explicitly acknowledged by the subscribing application, after it calls either **DDSDataReader::acknowledge_all** (p. 1565) or **DDSDataReader::acknowledge_sample** (p. 1564).

7.180.2.3 DDS_InstanceStateConsistencyKind

enum **DDS_InstanceStateConsistencyKind**

<<***extension***>> *(p. 213)* Whether instance state consistency is enabled.

QoS:

DDS_ReliabilityQosPolicy (p. 1295)

Enumerator

DDS_NO_RECOVER_INSTANCE_STATE_↩ CONSISTENCY	Instance state is not restored on a DataReader after reconnecting with a DataWriter until the DataWriter sends a new sample. When DataReaders rediscover DataWriters, they will not request updated instance state data. DataWriters always provide instance state data alongside each sample update regardless of this setting.
DDS_RECOVER_INSTANCE_STATE_CONSISTENCY	Instance state is restored on the DataReader after it reconnects with a DataWriter that has regained liveliness, even before the DataWriter sends a new sample. When DataReaders rediscover DataWriters, they will request updated instance state data. DataWriters will respond to requests for updated instance state data and publish updates on a builtin channel. DataWriters still provide instance state data alongside each sample update regardless of this setting. The following limitation applies to using RECOVER_INSTANCE_STATE_CONSISTENCY: If the **DDS_DestinationOrderQosPolicyKind** (p. 585) is set to **DDS_BY_SOURCE_TIMESTAMP_**↩ **DESTINATIONORDER_QOS** (p. 585), RECOVER_INSTANCE_STATE_CONSISTENCY can only be used if scope is set to **DDS_INSTANCE_**↩ **SCOPE_DESTINATIONORDER_QOS** (p. 586) (the default).

7.180.3 Variable Documentation

7.180.3.1 DDS_RELIABILITY_QOS_POLICY_NAME

```
const char* const DDS_RELIABILITY_QOS_POLICY_NAME  [extern]
```

Stringified human-readable name for **DDS_ReliabilityQosPolicy** (p. 1295).

7.181 RESOURCE_LIMITS

*<<**spec**>> (p. 209)* Controls the amount of physical memory allocated for DDS entities, if dynamic allocations are allowed, and how they occur. Also controls memory usage among different instance values for keyed topics.

Classes

- struct **DDS_ResourceLimitsQosPolicy**

  *<<**spec**>> (p. 209) Controls the amount of physical memory allocated for DDS entities, if dynamic allocations are allowed, and how they occur. Also controls memory usage among different instance values for keyed topics.*

Variables

- const char *const **DDS_RESOURCELIMITS_QOS_POLICY_NAME**

  *Stringified human-readable name for **DDS_ResourceLimitsQosPolicy** (p. 1308).*
- const **DDS_Long DDS_LENGTH_UNLIMITED**

  A special value indicating an unlimited quantity.

7.181.1 Detailed Description

*<<**spec**>> (p. 209)* Controls the amount of physical memory allocated for DDS entities, if dynamic allocations are allowed, and how they occur. Also controls memory usage among different instance values for keyed topics.

7.181.2 Variable Documentation

7.181.2.1 DDS_RESOURCELIMITS_QOS_POLICY_NAME

```
const char* const DDS_RESOURCELIMITS_QOS_POLICY_NAME  [extern]
```

Stringified human-readable name for **DDS_ResourceLimitsQosPolicy** (p. 1308).

7.181.2.2 DDS_LENGTH_UNLIMITED

```
const  DDS_Long DDS_LENGTH_UNLIMITED  [extern]
```

A special value indicating an unlimited quantity.

Examples

HelloWorld_subscriber.cxx.

Referenced by **connext::Requester< TReq, TRep >::read_replies()**, and **connext::Requester< TReq, TRep >::take_replies()**.

7.182 SERVICE

*<<**extension**>> (p. 213)* Service QoS policy is used to indicate what kind of service or tool is associated with the DDS entity.

Classes

- struct **DDS_ServiceQosPolicy**

  *<<**extension**>> (p. 213) Service or tool associated with a DDS entity.*

Enumerations

- enum **DDS_ServiceQosPolicyKind** {
  DDS_NO_SERVICE_QOS ,
  DDS_PERSISTENCE_SERVICE_QOS ,
  DDS_QUEUING_SERVICE_QOS ,
  DDS_ROUTING_SERVICE_QOS ,
  DDS_RECORDING_SERVICE_QOS ,
  DDS_REPLAY_SERVICE_QOS ,
  DDS_DATABASE_INTEGRATION_SERVICE_QOS ,
  DDS_WEB_INTEGRATION_SERVICE_QOS ,
  DDS_OBSERVABILITY_COLLECTOR_SERVICE_QOS ,
  DDS_ADMIN_CONSOLE_SERVICE_QOS }

  Kinds of service.

Variables

- const char ∗const **DDS_SERVICE_QOS_POLICY_NAME**

  *Stringified human-readable name for **DDS_ServiceQosPolicy** (p. 1358).*

7.182.1 Detailed Description

<<***extension***>> *(p. 213)* Service QoS policy is used to indicate what kind of service or tool is associated with the DDS entity.

7.182.2 Enumeration Type Documentation

7.182.2.1 DDS_ServiceQosPolicyKind

enum **DDS_ServiceQosPolicyKind**

Kinds of service.

QoS:

DDS_ServiceQosPolicy (p. 1358)

Enumerator

DDS_NO_SERVICE_QOS	There is no service associated with the entity.
DDS_PERSISTENCE_SERVICE_QOS	The entity is an entity created by RTI Persistence Service.
DDS_QUEUING_SERVICE_QOS	The entity is an entity created by RTI Queuing Service.
DDS_ROUTING_SERVICE_QOS	The entity is an entity created by RTI Routing Service.
DDS_RECORDING_SERVICE_QOS	The entity is an entity created by RTI Recording Service.
DDS_REPLAY_SERVICE_QOS	The entity is an entity created by RTI Replay Service.
DDS_DATABASE_INTEGRATION_SERVICE_QOS	The entity is an entity created by RTI Database Integration Service.
DDS_WEB_INTEGRATION_SERVICE_QOS	The entity is an entity created by RTI Web Integration Service.
DDS_OBSERVABILITY_COLLECTOR_SERVICE_QOS	The entity is an entity created by RTI Observability Collector Service.
DDS_ADMIN_CONSOLE_SERVICE_QOS	The entity is an entity created by RTI Admin Console.

7.182.3 Variable Documentation

7.182.3.1 DDS_SERVICE_QOS_POLICY_NAME

```
const char* const DDS_SERVICE_QOS_POLICY_NAME  [extern]
```

Stringified human-readable name for **DDS_ServiceQosPolicy** (p. 1358).

7.183 SYSTEM_RESOURCE_LIMITS

<<***extension***>> *(p. 213)* Configures DomainParticipant-independent resources used by RTI Connext.

Classes

- struct **DDS_SystemResourceLimitsQosPolicy**

  <<***extension***>> (p. *213) Configures* ***DDSDomainParticipant*** *(p. 1619)-independent resources used by RTI Connext. Mainly used to change the maximum number of* ***DDSDomainParticipant*** *(p. 1619) entities that can be created within a single process (address space).*

Variables

- const char ∗const **DDS_SYSTEMRESOURCELIMITS_QOS_POLICY_NAME**

  Stringified human-readable name for ***DDS_SystemResourceLimitsQosPolicy*** *(p. 1382).*

7.183.1 Detailed Description

<<***extension***>> *(p. 213)* Configures DomainParticipant-independent resources used by RTI Connext.

7.183.2 Variable Documentation

7.183.2.1 DDS_SYSTEMRESOURCELIMITS_QOS_POLICY_NAME

```
const char* const DDS_SYSTEMRESOURCELIMITS_QOS_POLICY_NAME  [extern]
```

Stringified human-readable name for **DDS_SystemResourceLimitsQosPolicy** (p. 1382).

7.184 TIME_BASED_FILTER

<<***spec***>> *(p. 209)* Filter that allows a **DDSDataReader** (p. 1556) to specify that it is interested only in (potentially) a subset of the values of the data.

Classes

- struct **DDS_TimeBasedFilterQosPolicy**

  <<***spec***>> *(p. 209) Filter that allows a **DDSDataReader** (p. 1556) to specify that it is interested only in (potentially) a subset of the values of the data.*

Variables

- const char ∗const **DDS_TIMEBASEDFILTER_QOS_POLICY_NAME**

  *Stringified human-readable name for **DDS_TimeBasedFilterQosPolicy** (p. 1389).*

7.184.1 Detailed Description

<<***spec***>> *(p. 209)* Filter that allows a **DDSDataReader** (p. 1556) to specify that it is interested only in (potentially) a subset of the values of the data.

7.184.2 Variable Documentation

7.184.2.1 DDS_TIMEBASEDFILTER_QOS_POLICY_NAME

```
const char* const DDS_TIMEBASEDFILTER_QOS_POLICY_NAME  [extern]
```

Stringified human-readable name for **DDS_TimeBasedFilterQosPolicy** (p. 1389).

7.185 TOPIC_DATA

Attaches a buffer of opaque data that is distributed by means of **Built-in Topics** (p. 61) during discovery.

Classes

- struct **DDS_TopicDataQosPolicy**

  *Attaches a buffer of opaque data that is distributed by means of **Built-in Topics** (p. 61) during discovery.*

Variables

- const char ∗const **DDS_TOPICDATA_QOS_POLICY_NAME**

  *Stringified human-readable name for **DDS_TopicDataQosPolicy** (p. 1396).*

7.185.1 Detailed Description

Attaches a buffer of opaque data that is distributed by means of **Built-in Topics** (p. 61) during discovery.

7.185.2 Variable Documentation

7.185.2.1 DDS_TOPICDATA_QOS_POLICY_NAME

```
const char* const DDS_TOPICDATA_QOS_POLICY_NAME  [extern]
```

Stringified human-readable name for **DDS_TopicDataQosPolicy** (p. 1396).

7.186 TOPIC_QUERY_DISPATCH

<<***extension***>> *(p. 213)* Configures the ability of a **DDSDataWriter** (p. 1589) to publish historical samples

Classes

- struct **DDS_TopicQueryDispatchQosPolicy**

  <<***extension***>> *(p. 213) Configures the ability of a **DDSDataWriter** (p. 1589) to publish samples in response to a **DDSTopicQuery** (p. 1895)*

Variables

- const char ∗const **DDS_TOPICQUERYDISPATCH_QOS_POLICY_NAME**

  *Stringified human-readable name for **DDS_TopicQueryDispatchQosPolicy** (p. 1404).*

7.186.1 Detailed Description

<<***extension***>> *(p. 213)* Configures the ability of a **DDSDataWriter** (p. 1589) to publish historical samples

7.186.2 Variable Documentation

7.186.2.1 DDS_TOPICQUERYDISPATCH_QOS_POLICY_NAME

```
const char* const DDS_TOPICQUERYDISPATCH_QOS_POLICY_NAME  [extern]
```

Stringified human-readable name for **DDS_TopicQueryDispatchQosPolicy** (p. 1404).

7.187 TRANSPORT_BUILTIN

<<***extension***>> *(p. 213)* Specifies which built-in transports are used.

Classes

- struct **DDS_TransportBuiltinQosPolicy**
  <<***extension***>> *(p. 213) Specifies which built-in transports are used.*

Macros

- #define **DDS_TRANSPORTBUILTIN_MASK_NONE**
  *None of the built-in transports will be registered automatically when the **DDSDomainParticipant** (p. 1619) is enabled.*
- #define **DDS_TRANSPORTBUILTIN_MASK_DEFAULT**
  *The default value of **DDS_TransportBuiltinQosPolicy::mask** (p. 1408).*
- #define **DDS_TRANSPORTBUILTIN_MASK_ALL**
  *All the available built-in transports are registered automatically when the **DDSDomainParticipant** (p. 1619) is enabled.*

Typedefs

- typedef **DDS_Long** **DDS_TransportBuiltinKindMask**
  *A mask of **DDS_TransportBuiltinKind** (p. 643) bits.*

Enumerations

- enum **DDS_TransportBuiltinKind** {
  DDS_TRANSPORTBUILTIN_UDPv4 ,
  DDS_TRANSPORTBUILTIN_SHMEM ,
  DDS_TRANSPORTBUILTIN_INTRA = 0x00000001 << 2 ,
  DDS_TRANSPORTBUILTIN_UDPv6 ,
  DDS_TRANSPORTBUILTIN_UDPv4_WAN }
  Built-in transport kind.

Variables

- const char ∗const **DDS_TRANSPORTBUILTIN_QOS_POLICY_NAME**

  *Stringified human-readable name for **DDS_TransportBuiltinQosPolicy** (p. 1408).*
- const char ∗const **DDS_TRANSPORTBUILTIN_SHMEM_ALIAS**

  Alias name for the shared memory built-in transport: "builtin.shmem".
- const char ∗const **DDS_TRANSPORTBUILTIN_UDPv4_ALIAS**

  Alias name for the UDPv4 built-in transport: "builtin.udpv4".
- const char ∗const **DDS_TRANSPORTBUILTIN_UDPv4_WAN_ALIAS**

  Alias name for the UDPv4 asymmetric built-in transport: "builtin.udpv4_wan".
- const char ∗const **DDS_TRANSPORTBUILTIN_UDPv6_ALIAS**

  Alias name for the UDPv6 built-in transport: "builtin.udpv6".

7.187.1 Detailed Description

<<***extension***>> *(p. 213)* Specifies which built-in transports are used.

See also

Changing the automatically registered built-in transports (p. 409)

7.187.2 Macro Definition Documentation

7.187.2.1 DDS_TRANSPORTBUILTIN_MASK_NONE

```
#define DDS_TRANSPORTBUILTIN_MASK_NONE
```

None of the built-in transports will be registered automatically when the **DDSDomainParticipant** (p. 1619) is enabled.

The user must explictly register transports using **NDDSTransportSupport::register_transport** (p. 2104).

See also

DDS_TransportBuiltinKindMask (p. 643)

7.187.2.2 DDS_TRANSPORTBUILTIN_MASK_DEFAULT

```
#define DDS_TRANSPORTBUILTIN_MASK_DEFAULT
```

The default value of **DDS_TransportBuiltinQosPolicy::mask** (p. 1408).

The set of builtin transport plugins that will be automatically registered with the participant by default. The user can register additional transports using **NDDSTransportSupport::register_transport** (p. 2104).

[default] [UDPv4|Shmem]

See also

DDS_TransportBuiltinKindMask (p. 643)

7.187.2.3 DDS_TRANSPORTBUILTIN_MASK_ALL

```
#define DDS_TRANSPORTBUILTIN_MASK_ALL
```

All the available built-in transports are registered automatically when the **DDSDomainParticipant** (p. 1619) is enabled.

See also

DDS_TransportBuiltinKindMask (p. 643)

7.187.3 Typedef Documentation

7.187.3.1 DDS_TransportBuiltinKindMask

```
typedef DDS_Long DDS_TransportBuiltinKindMask
```

A mask of **DDS_TransportBuiltinKind** (p. 643) bits.

QoS:

DDS_TransportBuiltinQosPolicy (p. 1408)

7.187.4 Enumeration Type Documentation

7.187.4.1 DDS_TransportBuiltinKind

```
enum DDS_TransportBuiltinKind
```

Built-in transport kind.

See also

DDS_TransportBuiltinKindMask (p. 643)

Enumerator

DDS_TRANSPORTBUILTIN_UDPv4	Built-in UDPv4 transport, ::**UDPv4 Transport** (p. 462).
DDS_TRANSPORTBUILTIN_SHMEM	Built-in shared memory transport, ::**Shared Memory Transport** (p. 456).
DDS_TRANSPORTBUILTIN_UDPv6	Built-in UDPv6 transport, ::**UDPv6 Transport** (p. 479).
DDS_TRANSPORTBUILTIN_UDPv4_WAN	Built-in UDPv4 asymmetric transport, ::**Real-Time WAN Transport** (p. 472).

7.187.5 Variable Documentation

7.187.5.1 DDS_TRANSPORTBUILTIN_QOS_POLICY_NAME

```
const char* const DDS_TRANSPORTBUILTIN_QOS_POLICY_NAME  [extern]
```

Stringified human-readable name for **DDS_TransportBuiltinQosPolicy** (p. 1408).

7.187.5.2 DDS_TRANSPORTBUILTIN_SHMEM_ALIAS

```
const char* const DDS_TRANSPORTBUILTIN_SHMEM_ALIAS  [extern]
```

Alias name for the shared memory built-in transport: "builtin.shmem".

7.187.5.3 DDS_TRANSPORTBUILTIN_UDPv4_ALIAS

```
const char* const DDS_TRANSPORTBUILTIN_UDPv4_ALIAS  [extern]
```

Alias name for the UDPv4 built-in transport: "builtin.udpv4".

7.187.5.4 DDS_TRANSPORTBUILTIN_UDPv4_WAN_ALIAS

```
const char* const DDS_TRANSPORTBUILTIN_UDPv4_WAN_ALIAS  [extern]
```

Alias name for the UDPv4 asymmetric built-in transport: "builtin.udpv4_wan".

7.187.5.5 DDS_TRANSPORTBUILTIN_UDPv6_ALIAS

```
const char* const DDS_TRANSPORTBUILTIN_UDPv6_ALIAS  [extern]
```

Alias name for the UDPv6 built-in transport: "builtin.udpv6".

7.188 TRANSPORT_MULTICAST

<<***extension***>> *(p. 213)* Specifies the multicast address on which a **DDSDataReader** (p. 1556) wants to receive its data. It can also specify a port number, as well as a subset of the available (at the **DDSDomainParticipant** (p. 1619) level) transports with which to receive the multicast data.

Modules

- **Multicast Settings**

  <<***extension***>> *(p. 213) Multicast communication settings.*
- **Multicast Mapping**

  <<***extension***>> *(p. 213) Multicast communication mapping.*

Classes

- struct **DDS_TransportMulticastQosPolicy**

  <<***extension***>> *(p. 213) Specifies the multicast address on which a **DDSDataReader** (p. 1556) wants to receive its data. It can also specify a port number as well as a subset of the available (at the **DDSDomainParticipant** (p. 1619) level) transports with which to receive the multicast data.*

Enumerations

- enum **DDS_TransportMulticastQosPolicyKind** {
  DDS_AUTOMATIC_TRANSPORT_MULTICAST_QOS ,
  DDS_UNICAST_ONLY_TRANSPORT_MULTICAST_QOS }

  Transport Multicast Policy Kind.

Variables

- const char ∗const **DDS_TRANSPORTMULTICAST_QOS_POLICY_NAME**

  *Stringified human-readable name for **DDS_TransportMulticastQosPolicy** (p. 1415).*

7.188.1 Detailed Description

<<***extension***>> *(p. 213)* Specifies the multicast address on which a **DDSDataReader** (p. 1556) wants to receive its data. It can also specify a port number, as well as a subset of the available (at the **DDSDomainParticipant** (p. 1619) level) transports with which to receive the multicast data.

7.188.2 Enumeration Type Documentation

7.188.2.1 DDS_TransportMulticastQosPolicyKind

```
enum DDS_TransportMulticastQosPolicyKind
```

Transport Multicast Policy Kind.

See also

DDS_TransportMulticastQosPolicy (p. 1415)

Enumerator

DDS_AUTOMATIC_TRANSPORT_MULTICAST_QOS	Selects the multicast address automatically. NOTE: This setting is required when using the **DDS_TransportMulticastMappingQosPolicy** (p. 1413).
DDS_UNICAST_ONLY_TRANSPORT_MULTICAST_↩ QOS	Selects a unicast-only mode.

7.188.3 Variable Documentation

7.188.3.1 DDS_TRANSPORTMULTICAST_QOS_POLICY_NAME

```
const char* const DDS_TRANSPORTMULTICAST_QOS_POLICY_NAME  [extern]
```

Stringified human-readable name for **DDS_TransportMulticastQosPolicy** (p. 1415).

7.189 TRANSPORT_MULTICAST_MAPPING

<<***extension***>> *(p. 213)* Specifies a list of topic expressions and addresses that can be used by an Entity with a specific topic name to receive data.

Classes

- struct **DDS_TransportMulticastMappingQosPolicy**

  <<***extension***>> *(p. 213) Specifies a list of topic_expressions and multicast addresses that can be used by an Entity with a specific topic name to receive data.*

Variables

- const char ∗const **DDS_TRANSPORTMULTICASTMAPPING_QOS_POLICY_NAME**

  *Stringified human-readable name for **DDS_TransportMulticastMappingQosPolicy** (p. 1413).*

7.189.1 Detailed Description

<<***extension***>> *(p. 213)* Specifies a list of topic expressions and addresses that can be used by an Entity with a specific topic name to receive data.

7.189.2 Variable Documentation

7.189.2.1 DDS_TRANSPORTMULTICASTMAPPING_QOS_POLICY_NAME

```
const char* const DDS_TRANSPORTMULTICASTMAPPING_QOS_POLICY_NAME  [extern]
```

Stringified human-readable name for **DDS_TransportMulticastMappingQosPolicy** (p. 1413).

7.190 TRANSPORT_PRIORITY

<<***spec***>> *(p. 209)* This QoS policy allows the application to take advantage of transport that are capable of sending messages with different priorities.

Classes

- struct **DDS_TransportPriorityQosPolicy**

  *<<**spec**>> (p. 209) This QoS policy allows the application to take advantage of transports that are capable of sending messages with different priorities.*

Variables

- const char ∗const **DDS_TRANSPORTPRIORITY_QOS_POLICY_NAME**

  *Stringified human-readable name for **DDS_TransportPriorityQosPolicy** (p. 1419).*

7.190.1 Detailed Description

<<***spec***>> *(p. 209)* This QoS policy allows the application to take advantage of transport that are capable of sending messages with different priorities.

7.190.2 Variable Documentation

7.190.2.1 DDS_TRANSPORTPRIORITY_QOS_POLICY_NAME

```
const char* const DDS_TRANSPORTPRIORITY_QOS_POLICY_NAME  [extern]
```

Stringified human-readable name for **DDS_TransportPriorityQosPolicy** (p. 1419).

7.191 TRANSPORT_SELECTION

<<***extension***>> *(p. 213)* Specifies the physical transports that a **DDSDataWriter** (p. 1589) or **DDSDataReader** (p. 1556) may use to send or receive data.

Classes

- struct **DDS_TransportSelectionQosPolicy**

  <<***extension***>> *(p. 213) Specifies the physical transports a **DDSDataWriter** (p. 1589) or **DDSDataReader** (p. 1556) may use to send or receive data.*

Variables

- const char *const **DDS_TRANSPORTSELECTION_QOS_POLICY_NAME**

  *Stringified human-readable name for **DDS_TransportSelectionQosPolicy** (p. 1421).*

7.191.1 Detailed Description

<<***extension***>> *(p. 213)* Specifies the physical transports that a **DDSDataWriter** (p. 1589) or **DDSDataReader** (p. 1556) may use to send or receive data.

7.191.2 Variable Documentation

7.191.2.1 DDS_TRANSPORTSELECTION_QOS_POLICY_NAME

```
const char* const DDS_TRANSPORTSELECTION_QOS_POLICY_NAME  [extern]
```

Stringified human-readable name for **DDS_TransportSelectionQosPolicy** (p. 1421).

7.192 TRANSPORT_UNICAST

<<***extension***>> *(p. 213)* Specifies a subset of transports and a port number that can be used by an Entity to receive data.

Modules

- **Unicast Settings**

  <<***extension***>> *(p. 213) Unicast communication settings.*

Classes

- struct **DDS_TransportUnicastQosPolicy**

  <<***extension***>> *(p. 213) Specifies a subset of transports and a port number that can be used by an Entity to receive data.*

Variables

- const char ∗const **DDS_TRANSPORTUNICAST_QOS_POLICY_NAME**

  *Stringified human-readable name for **DDS_TransportUnicastQosPolicy** (p. 1422).*

7.192.1 Detailed Description

<<***extension***>> *(p. 213)* Specifies a subset of transports and a port number that can be used by an Entity to receive data.

7.192.2 Variable Documentation

7.192.2.1 DDS_TRANSPORTUNICAST_QOS_POLICY_NAME

```
const char* const DDS_TRANSPORTUNICAST_QOS_POLICY_NAME  [extern]
```

Stringified human-readable name for **DDS_TransportUnicastQosPolicy** (p. 1422).

7.193 TYPE_CONSISTENCY_ENFORCEMENT

<<***spec***>> *(p. 209)* Defines the rules for determining whether the type used to publish a given topic is consistent with that used to subscribe to it.

Classes

- struct **DDS_TypeConsistencyEnforcementQosPolicy**

  *<<**spec**>> (p. 209) Defines the rules for determining whether the type used to publish a given topic is consistent with that used to subscribe to it.*

Enumerations

- enum **DDS_TypeConsistencyKind** {
  DDS_DISALLOW_TYPE_COERCION ,
  DDS_ALLOW_TYPE_COERCION ,
  DDS_AUTO_TYPE_COERCION }

  Kinds of type consistency.

Variables

- const char *const **DDS_TYPE_CONSISTENCY_ENFORCEMENT_QOS_POLICY_NAME**

  *Stringified human-readable name for **DDS_TypeConsistencyEnforcementQosPolicy** (p. 1493).*

7.193.1 Detailed Description

<<**spec**>> (p. 209) Defines the rules for determining whether the type used to publish a given topic is consistent with that used to subscribe to it.

7.193.2 Enumeration Type Documentation

7.193.2.1 DDS_TypeConsistencyKind

`enum` **DDS_TypeConsistencyKind**

Kinds of type consistency.

QoS:

DDS_TypeConsistencyEnforcementQosPolicy (p. 1493)

Enumerator

DDS_DISALLOW_TYPE_COERCION	The DataWriter and the DataReader must support the same data type in order for them to communicate. This is the degree of type consistency enforcement required by the OMG DDS Specification prior to the OMG Extensible and Dynamic Topic Types for DDS Specification.

DDS_ALLOW_TYPE_COERCION	The DataWriter and the DataReader need not support the same data type in order for them to communicate as long as the DataReader's type is assignable from the DataWriter's type. For example, the following two extensible types will be assignable to each other since MyDerivedType contains all the members of MyBaseType (member_1) plus some additional elements (member_2).
	`struct MyBaseType { long member_1; };`
	`struct MyDerivedType: MyBaseType { long member_2; };`
	Even if MyDerivedType was not explicitly inheriting from MyBaseType the types would still be assignable. For example:
	`struct MyBaseType { long member_1; };`
	`struct MyDerivedType { long member_1; long member_2; };`
	For additional information on type assignability refer to the OMG Extensible and Dynamic Topic Types for DDS Specification.

DDS_AUTO_TYPE_COERCION	This AUTO value will be applied as **DDS_DISALLOW_TYPE_COERCION** (p. 651) when the data type is annotated with @transfer_mode(SHMEM_REF) and the @language_binding is PLAIN (default) while using C, Traditional C++, or Modern C++ APIs. In all other cases, this AUTO value will be applied as **DDS_ALLOW_TYPE_COERCION** (p. 651). **[default]**

7.193.3 Variable Documentation

7.193.3.1 DDS_TYPE_CONSISTENCY_ENFORCEMENT_QOS_POLICY_NAME

```
const char* const DDS_TYPE_CONSISTENCY_ENFORCEMENT_QOS_POLICY_NAME  [extern]
```

Stringified human-readable name for **DDS_TypeConsistencyEnforcementQosPolicy** (p. 1493).

7.194 TYPESUPPORT

[DEPRECATED] (p. **384)** <<***extension***>> *(*p. *213)* Allows you to attach application-specific values to a **DDSData↩ Writer** (p. 1589) or **DDSDataReader** (p. 1556), which are passed to the serialization or deserialization routine of the associated data type and choose whether to set padding bytes to zero during serialization.

Classes

- struct **DDS_TypeSupportQosPolicy**

  [DEPRECATED] (p. ***384***) *<<**extension**>> (p. 213) Allows you to attach application-specific values to a **DDSDataWriter** (p. 1589) or **DDSDataReader** (p. 1556), which are passed to the serialization or deserialization routine of the associated data type and choose whether to set padding bytes to zero during serialization.*

Enumerations

- enum **DDS_CdrPaddingKind** {
  DDS_ZERO_CDR_PADDING ,
  DDS_NOT_SET_CDR_PADDING ,
  DDS_AUTO_CDR_PADDING }

  The CDR padding kind determines whether or not the padding bytes will be set to zero during CDR serialization.

Variables

- const char ∗const **DDS_TYPESUPPORT_QOS_POLICY_NAME**

  *Stringified human-readable name for **DDS_TypeSupportQosPolicy** (p. 1499).*

7.194.1 Detailed Description

[DEPRECATED] (p. **384**) *<<**extension**>> (p. 213)* Allows you to attach application-specific values to a **DDSDataWriter** (p. 1589) or **DDSDataReader** (p. 1556), which are passed to the serialization or deserialization routine of the associated data type and choose whether to set padding bytes to zero during serialization.

7.194.2 Enumeration Type Documentation

7.194.2.1 DDS_CdrPaddingKind

enum **DDS_CdrPaddingKind**

The CDR padding kind determines whether or not the padding bytes will be set to zero during CDR serialization.

Enumerator

DDS_ZERO_CDR_PADDING	Padding bytes will be set to zeros during CDR serialization.
DDS_NOT_SET_CDR_PADDING	Padding bytes will not be set to any value during CDR serialization.
DDS_AUTO_CDR_PADDING	When set on a **DDSDomainParticipant** (p. 1619) the default behavior is **DDS_NOT_SET_CDR_PADDING** (p. 652). When set on a **DDSDataWriter** (p. 1589) or **DDSDataReader** (p. 1556) the behavior is to inherit the value from the **DDSDomainParticipant** (p. 1619).

7.194.3 Variable Documentation

7.194.3.1 DDS_TYPESUPPORT_QOS_POLICY_NAME

```
const char* const DDS_TYPESUPPORT_QOS_POLICY_NAME  [extern]
```

Stringified human-readable name for **DDS_TypeSupportQosPolicy** (p. 1499).

7.195 USER_DATA

Attaches a buffer of opaque data that is distributed by means of **Built-in Topics** (p. 61) during discovery.

Classes

- struct **DDS_UserDataQosPolicy**
  *Attaches a buffer of opaque data that is distributed by means of **Built-in Topics** (p. 61) during discovery.*

Variables

- const char *const **DDS_USERDATA_QOS_POLICY_NAME**
  *Stringified human-readable name for **DDS_UserDataQosPolicy** (p. 1504).*

7.195.1 Detailed Description

Attaches a buffer of opaque data that is distributed by means of **Built-in Topics** (p. 61) during discovery.

7.195.2 Variable Documentation

7.195.2.1 DDS_USERDATA_QOS_POLICY_NAME

```
const char* const DDS_USERDATA_QOS_POLICY_NAME  [extern]
```

Stringified human-readable name for **DDS_UserDataQosPolicy** (p. 1504).

7.196 WRITER_DATA_LIFECYCLE

<<***spec***>> *(p. 209)* Controls how a DataWriter handles the lifecycle of the instances (keys) that it is registered to manage.

Classes

- struct **DDS_WriterDataLifecycleQosPolicy**

  <<***spec***>> *(p. 209) Controls how a **DDSDataWriter** (p. 1589) handles the lifecycle of the instances (keys) that it is registered to manage.*

Variables

- const char ∗const **DDS_WRITERDATALIFECYCLE_QOS_POLICY_NAME**

  *Stringified human-readable name for **DDS_WriterDataLifecycleQosPolicy** (p. 1524).*

7.196.1 Detailed Description

<<***spec***>> *(p. 209)* Controls how a DataWriter handles the lifecycle of the instances (keys) that it is registered to manage.

7.196.2 Variable Documentation

7.196.2.1 DDS_WRITERDATALIFECYCLE_QOS_POLICY_NAME

```
const char* const DDS_WRITERDATALIFECYCLE_QOS_POLICY_NAME  [extern]
```

Stringified human-readable name for **DDS_WriterDataLifecycleQosPolicy** (p. 1524).

7.197 WIRE_PROTOCOL

<<***extension***>> *(p. 213)* Specifies the wire protocol related attributes for the **DDSDomainParticipant** (p. 1619).

Classes

- struct **DDS_RtpsWellKnownPorts_t**

  <<***extension***>> *(p. 213) RTPS well-known port mapping configuration.*
- struct **DDS_WireProtocolQosPolicy**

  <<***extension***>> *(p. 213) Specifies the wire-protocol-related attributes for the **DDSDomainParticipant** (p. 1619).*

Macros

- #define **DDS_RTPS_RESERVED_PORT_MASK_DEFAULT**

  *The default value of **DDS_WireProtocolQosPolicy::rtps_reserved_port_mask** (p. 1517).*
- #define **DDS_RTPS_RESERVED_PORT_MASK_NONE**

  No bits are set.
- #define **DDS_RTPS_RESERVED_PORT_MASK_ALL**

  All bits are set.

Typedefs

- typedef **DDS_Long DDS_RtpsReservedPortKindMask**

  *A mask of **DDS_RtpsReservedPortKind** (p. 657) bits.*

Enumerations

- enum **DDS_RtpsReservedPortKind** {
  DDS_RTPS_RESERVED_PORT_BUILTIN_UNICAST = 0x0001 << 0 ,
  DDS_RTPS_RESERVED_PORT_BUILTIN_MULTICAST = 0x0001 << 1 ,
  DDS_RTPS_RESERVED_PORT_USER_UNICAST = 0x0001 << 2 ,
  DDS_RTPS_RESERVED_PORT_USER_MULTICAST = 0x0001 << 3 }

  RTPS reserved port kind, used to identify the types of ports that can be reserved on domain participant enable.
- enum **DDS_WireProtocolQosPolicyAutoKind** {
  DDS_RTPS_AUTO_ID_FROM_IP = 0 ,
  DDS_RTPS_AUTO_ID_FROM_MAC = 1 ,
  DDS_RTPS_AUTO_ID_FROM_UUID = 2 }

  Mechanism to automatically calculate the GUID prefix.

Variables

- const struct **DDS_RtpsWellKnownPorts_t DDS_RTI_BACKWARDS_COMPATIBLE_RTPS_WELL_↩ KNOWN_PORTS**

  Assign to use well-known port mappings which are compatible with previous versions of the RTI Connext middleware.
- const struct **DDS_RtpsWellKnownPorts_t DDS_INTEROPERABLE_RTPS_WELL_KNOWN_PORTS**

  Assign to use well-known port mappings which are compliant with OMG's DDS Interoperability Wire Protocol.
- const char *const **DDS_WIREPROTOCOL_QOS_POLICY_NAME**

  *Stringified human-readable name for **DDS_WireProtocolQosPolicy** (p. 1511).*

7.197.1 Detailed Description

<<***extension***>> *(p. 213)* Specifies the wire protocol related attributes for the **DDSDomainParticipant** (p. 1619).

7.197.2 Macro Definition Documentation

7.197.2.1 DDS_RTPS_RESERVED_PORT_MASK_DEFAULT

```
#define DDS_RTPS_RESERVED_PORT_MASK_DEFAULT
```

Value:

```
((DDS_RtpsReservedPortKindMask) DDS_RTPS_RESERVED_PORT_BUILTIN_UNICAST \
 | DDS_RTPS_RESERVED_PORT_BUILTIN_MULTICAST | DDS_RTPS_RESERVED_PORT_USER_UNICAST)
```

The default value of **DDS_WireProtocolQosPolicy::rtps_reserved_port_mask** (p. 1517).

Most of the ports that may be needed by DDS will be reserved by the transport when the participant is enabled. With this value set, failure to allocate a port that is computed based on the **DDS_RtpsWellKnownPorts_t** (p. 1338) will be detected at this time and the enable operation will fail.

This setting will avoid reserving the **usertraffic** multicast port, which is not actually used unless there are DataReaders that enable multicast but fail to specify a port.

Automatic participant ID selection will be based on finding a participant index with both the discovery (metatraffic) unicast port and usertraffic unicast port available.

See also

DDS_RtpsReservedPortKindMask (p. 657)

7.197.2.2 DDS_RTPS_RESERVED_PORT_MASK_NONE

```
#define DDS_RTPS_RESERVED_PORT_MASK_NONE
```

No bits are set.

None of the ports that are needed by DDS will be allocated until they are specifically required. With this value set, automatic participant Id selection will be based on selecting a port for discovery (metatraffic) unicast traffic on a single transport.

See also

DDS_RtpsReservedPortKindMask (p. 657)

7.197.2.3 DDS_RTPS_RESERVED_PORT_MASK_ALL

```
#define DDS_RTPS_RESERVED_PORT_MASK_ALL
```

All bits are set.

All of the ports that may be needed by DDS will be reserved when the participant is enabled. With this value set, failure to allocate a port that is computed based on the **DDS_RtpsWellKnownPorts_t** (p. 1338) will be detected at this time, and the enable operation will fail.

Note that this will also reserve the **usertraffic** multicast port which is not actually used unless there are DataReaders that enable multicast but fail to specify a port. To avoid unnecesary resource usage for these ports, use RTPS_↩ RESERVED_PORT_MASK_DEFAULT.

Automatic participant ID selection will be based on finding a participant index with both the discovery (metatraffic) unicast port and usertraffic unicast port available.

See also

DDS_RtpsReservedPortKindMask (p. 657)

7.197.3 Typedef Documentation

7.197.3.1 DDS_RtpsReservedPortKindMask

```
typedef DDS_Long DDS_RtpsReservedPortKindMask
```

A mask of **DDS_RtpsReservedPortKind** (p. 657) bits.

QoS:

DDS_WireProtocolQosPolicy (p. 1511)

7.197.4 Enumeration Type Documentation

7.197.4.1 DDS_RtpsReservedPortKind

```
enum DDS_RtpsReservedPortKind
```

RTPS reserved port kind, used to identify the types of ports that can be reserved on domain participant enable.

See also

DDS_WireProtocolQosPolicy::rtps_reserved_port_mask (p. 1517)

Enumerator

DDS_RTPS_RESERVED_PORT_BUILTIN_UNICAST	Select the **metatraffic** unicast port.
DDS_RTPS_RESERVED_PORT_BUILTIN_MULTICAST	Select the **metatraffic** multicast port.
DDS_RTPS_RESERVED_PORT_USER_UNICAST	Select the **usertraffic** unicast port.
DDS_RTPS_RESERVED_PORT_USER_MULTICAST	Select the **usertraffic** multicast port.

7.197.4.2 DDS_WireProtocolQosPolicyAutoKind

`enum` **`DDS_WireProtocolQosPolicyAutoKind`**

Mechanism to automatically calculate the GUID prefix.

See also

DDS_WireProtocolQosPolicy::rtps_auto_id_kind (p. 1518)

Enumerator

DDS_RTPS_AUTO_ID_FROM_IP	Builds the GUID prefix using the IPv4 address and process ID. Uses the IPv4 address of the first up-and-running interface of the host machine and the process ID to build the GUID prefix.
DDS_RTPS_AUTO_ID_FROM_MAC	Builds the GUID prefix using the MAC address and process ID. Uses the first 32 bits of the MAC address assigned to the first up-and-running interface and the process ID to build the GUID prefix. Note to Android Users: DDS_RTPS_AUTO_ID_FROM_MAC is not supported in recent versions of Android (6.0 and later).
DDS_RTPS_AUTO_ID_FROM_UUID	Builds the GUID prefix by selecting the UUID-based algorithm (default). This method of generating the GUID prefix does not require having a network interface and is friendly to IP mobility scenarios in which an RTI Connext application may start in a node that does not have a physical network interface enabled.

7.197.5 Variable Documentation

7.197.5.1 DDS_RTI_BACKWARDS_COMPATIBLE_RTPS_WELL_KNOWN_PORTS

`const struct` **`DDS_RtpsWellKnownPorts_t`** `DDS_RTI_BACKWARDS_COMPATIBLE_RTPS_WELL_KNOWN_PORTS [extern]`

Assign to use well-known port mappings which are compatible with previous versions of the RTI Connext middleware.

Assign **DDS_WireProtocolQosPolicy::rtps_well_known_ports** (p. 1517) to this value to remain compatible with previous versions of the RTI Connext middleware that used fixed port mappings.

The following are the `rtps_well_known_ports` values for **DDS_RTI_BACKWARDS_COMPATIBLE_RTPS_↩ WELL_KNOWN_PORTS** (p. 658):

`port_base` = 7400
`domain_id_gain` = 10
`participant_id_gain` = 1000
`builtin_multicast_port_offset` = 2
`builtin_unicast_port_offset` = 0
`user_multicast_port_offset` = 1
`user_unicast_port_offset` = 3

These settings are *not* compliant with OMG's DDS Interoperability Wire Protocol. To comply with the specification, please use **DDS_INTEROPERABLE_RTPS_WELL_KNOWN_PORTS** (p. 659).

See also

DDS_WireProtocolQosPolicy::rtps_well_known_ports (p. 1517)

DDS_INTEROPERABLE_RTPS_WELL_KNOWN_PORTS (p. 659)

7.197.5.2 DDS_INTEROPERABLE_RTPS_WELL_KNOWN_PORTS

`const struct` **DDS_RtpsWellKnownPorts_t** `DDS_INTEROPERABLE_RTPS_WELL_KNOWN_PORTS [extern]`

Assign to use well-known port mappings which are compliant with OMG's DDS Interoperability Wire Protocol.

Assign **DDS_WireProtocolQosPolicy::rtps_well_known_ports** (p. 1517) to this value to use well-known port mappings which are compliant with OMG's DDS Interoperability Wire Protocol.

The following are the `rtps_well_known_ports` values for **DDS_INTEROPERABLE_RTPS_WELL_KNOWN_↩ PORTS** (p. 659):

`port_base` = 7400
`domain_id_gain` = 250
`participant_id_gain` = 2
`builtin_multicast_port_offset` = 0
`builtin_unicast_port_offset` = 10
`user_multicast_port_offset` = 1
`user_unicast_port_offset` = 11

Assuming a maximum port number of 65535 (UDPv4), the above settings enable the use of about 230 domains with up to 120 Participants per node per domain.

These settings are *not* backwards compatible with previous versions of the RTI Connext middleware that used fixed port mappings. For backwards compability, please use **DDS_RTI_BACKWARDS_COMPATIBLE_RTPS_WELL_KNOWN↩ _PORTS** (p. 658).

See also

DDS_WireProtocolQosPolicy::rtps_well_known_ports (p. 1517)

DDS_RTI_BACKWARDS_COMPATIBLE_RTPS_WELL_KNOWN_PORTS (p. 658)

7.197.5.3 DDS_WIREPROTOCOL_QOS_POLICY_NAME

```
const char* const DDS_WIREPROTOCOL_QOS_POLICY_NAME  [extern]
```

Stringified human-readable name for **DDS_WireProtocolQosPolicy** (p. 1511).

7.198 Extended Qos Support

<<***extension***>> *(p. 213)* Types and defines used in extended QoS policies.

Modules

- **Thread Settings**

  The properties of a thread of execution. Consult `Platform Notes` *for additional platform specific details.*

Classes

- struct **DDS_RtpsReliableReaderProtocol_t**

  <<***extension***>> *(p. 213) Qos related to reliable reader protocol defined in RTPS.*
- struct **DDS_RtpsReliableWriterProtocol_t**

  <<***extension***>> *(p. 213) QoS related to the reliable writer protocol defined in RTPS.*

7.198.1 Detailed Description

<<***extension***>> *(p. 213)* Types and defines used in extended QoS policies.

7.199 Unicast Settings

<<***extension***>> *(p. 213)* Unicast communication settings.

Classes

- struct **DDS_TransportUnicastSettings_t**

  Type representing a list of unicast locators.
- struct **DDS_TransportUnicastSettingsSeq**

  Declares IDL `sequence<` ***DDS_TransportUnicastSettings_t*** *(p. 1424)* `>`

7.199.1 Detailed Description

<<***extension***>> *(p. 213)* Unicast communication settings.

7.200 Multicast Settings

<<***extension***>> *(*p. *213)* Multicast communication settings.

Classes

- struct **DDS_TransportMulticastSettings_t**

  Type representing a list of multicast locators.
- struct **DDS_TransportMulticastSettingsSeq**

  Declares IDL `sequence<` ***DDS_TransportMulticastSettings_t*** *(*p. *1417)* `>`

7.200.1 Detailed Description

<<***extension***>> *(*p. *213)* Multicast communication settings.

7.201 Multicast Mapping

<<***extension***>> *(*p. *213)* Multicast communication mapping.

Classes

- struct **DDS_TransportMulticastMappingFunction_t**

  Type representing an external mapping function.
- struct **DDS_TransportMulticastMapping_t**

  <<***extension***>> *(*p. *213) Type representing a list of multicast mapping elements.*
- struct **DDS_TransportMulticastMappingSeq**

  Declares IDL `sequence<` ***DDS_TransportMulticastMapping_t*** *(*p. *1410)* `>`

7.201.1 Detailed Description

<<***extension***>> *(*p. *213)* Multicast communication mapping.

7.202 NDDS_DISCOVERY_PEERS

Environment variable or a file that specifies the default values of **DDS_DiscoveryQosPolicy::initial_peers** (p. 962) and **DDS_DiscoveryQosPolicy::multicast_receive_addresses** (p. 962) contained in the **DDS_DomainParticipantQos**↩ **::discovery** (p. 974) qos policy.

Environment variable or a file that specifies the default values of **DDS_DiscoveryQosPolicy::initial_peers** (p. 962) and **DDS_DiscoveryQosPolicy::multicast_receive_addresses** (p. 962) contained in the **DDS_DomainParticipantQos**↩ **::discovery** (p. 974) qos policy.

The default value of the **DDS_DomainParticipantQos** (p. 971) is obtained by calling **DDSDomainParticipantFactory**↩ **::get_default_participant_qos()** (p. 1700).

NDDS_DISCOVERY_PEERS specifies the default value of the **DDS_DiscoveryQosPolicy::initial_peers** (p. 962) and **DDS_DiscoveryQosPolicy::multicast_receive_addresses** (p. 962) fields, when the default participant QoS policies have not been explictly set by the user (i.e., **DDSDomainParticipantFactory::set_default_participant_qos()** (p. 1698) has never been called or was called using **DDS_PARTICIPANT_QOS_DEFAULT** (p. 50)).

If NDDS_DISCOVERY_PEERS does *not* contain a multicast address, then the string sequence **DDS_DiscoveryQos**↩ **Policy::multicast_receive_addresses** (p. 962) is cleared and the RTI discovery process will not listen for discovery messages via multicast.

If NDDS_DISCOVERY_PEERS contains one or more multicast addresses, the addresses will be stored in **DDS_**↩ **DiscoveryQosPolicy::multicast_receive_addresses** (p. 962), starting at element 0. They will be stored in the order in which they appear in NDDS_DISCOVERY_PEERS.

Note: IPv4 multicast addresses must have a prefix. Therefore, when using the UDPv6 transport: if there are any IPv4 multicast addresses in the peers list, make sure they have "udpv4://" in front of them (such as udpv4://239.255.0.1).

Note: Currently, RTI Connext will only listen for discovery traffic on the first multicast address (element 0) in **DDS_**↩ **DiscoveryQosPolicy::multicast_receive_addresses** (p. 962).

NDDS_DISCOVERY_PEERS provides a mechanism to dynamically switch the discovery configuration of an RTI Connext application without recompilation. The application programmer is free to not use the default values; instead use values supplied by other means.

NDDS_DISCOVERY_PEERS can be specified either in an environment variable as comma (',') separated "peer descriptors" (see **Peer Descriptor Format** (p. 663)) or in a file. These formats are described below.

7.202.1 Peer Descriptor Format

A **peer descriptor** string specifies a range of participants at a given `locator`. Peer descriptor strings are used in the **DDS_DiscoveryQosPolicy::initial_peers** (p. 962) field and the **DDSDomainParticipant::add_peer()** (p. 1678) operation.

The anatomy of a `peer` descriptor is illustrated below using a UDPv4 transport and a custom "StarFabric" transport example.

A peer descriptor consists of:

optional **Participant ID Limit**. If a simple integer is specified, it indicates the maximum participant ID to be contacted by the RTI Connext discovery mechanism at the given locator. If that integer is enclosed in square brackets (e.g.: [2]) *only* that Participant ID will be used. You can also specify a range in the form of [a-b]: in this case only the Participant IDs in that specific range are contacted. If omitted, a default value of 4 is implied: participant IDs 0,1,2,3, and 4 will be contacted.

- **Locator**. See **Locator Format** (p. 663).

These are separated by the '@' character. The separator may be omitted if a participant ID limit is not explictly specified.

Note that the "participant ID limit" only applies to unicast locators; it is ignored for multicast locators (and therefore should be omitted for multicast peer descriptors).

7.202.1.1 Locator Format

A **locator** string specifies a transport and an address in string format. Locators are used to form peer descriptors. A locator is equivalent to a peer descriptor with the default maximum participant ID.

A locator consists of:

optional **Transport name** (**alias** or class). This identifies the set of transport plugins (**Transport Aliases** (p. 173)) that may be used to parse the `address` portion of the locator. Note that a transport class name is an implicit alias that is used to refer to all the transport plugin instances of that class.

optional **Address**. See **Address Format** (p. 664).

These are separated by the "//:" string. The separator is specified if and only if a transport name is specified.

If a transport name is specified, the address may be omitted; in that case, all the unicast addresses (across all transport plugin instances) associated with the transport class are implied. Thus, a locator string may specify several addresses.

If an address is specified, the transport name and the separator string may be omitted; in that case all the available transport plugins (for the **DDSEntity** (p. 1731)) may be used to parse the address string.

7.202.1.2 Address Format

An **address** string specifies a transport-independent network address that qualifies a **transport-dependent** address string. Addresses are used to form locators. Addresses are also used in **DDS_DiscoveryQosPolicy::multicast_receive_addresses** (p. 962), and **DDS_TransportMulticastSettings_t::receive_address** (p. 1418) fields. An address is equivalent to a locator in which the transport name and separator are omitted.

An address consists of:

optional **Network Address**. An address in IPv4 or IPv6 string notation. If omitted, the network address of the transport is implied (**Transport Network Address** (p. 176)).

optional **Transport Address**. A string that is passed to the transport for processing. The transport maps this string into **NDDS_Transport_Property_t::address_bit_count** (p. 2046) bits. If omitted the network address is used as the fully qualified address.

These are separated by the '#' character. If a separator is specified, it must be followed by a non-empty string which is passed to the transport plugin. If the separator is omitted, it is treated as a transport address with an implicit network address (of the transport plugin). The implicit network address is the address used when registering the transport: e.g, the UDPv4 implicit network address is 0.0.0.0.0.0.0.0.0.0.0.0.

The bits resulting from the transport address string are combined with the network address. The least or most significant **NDDS_Transport_Property_t::address_bit_count** (p. 2046) bits of the network address are ignored and replaced with the transport address. The sign of the **NDDS_Transport_Property_t::address_bit_count** (p. 2046) value determines whether the most significant bits (negative sign) or least significant bits (positive sign) are ignored (**Transport Network Address** (p. 176)).

You can use a hostname instead of a transport address. If a hostname is used, the transport will attempt to resolve the hostname upon enabling the DomainParticipant. Changes to the hostname resolution will only be detected if the DNS tracker is enabled (it is enabled by default); see `Using DNS Tracker to Keep Peer List Updated`, in the Core Libraries User's Manual. For example:

```
<domain_participant_qos>

    <discovery>

        <initial_peers>

            <element>udpv4://myhostname</element>

        </initial_peers>

    </discovery>

</domain_participant_qos>
```

In this example, the DomainParticipant will attempt to resolve "myhostname" using the UDPv4 transport.

7.202.2 NDDS_DISCOVERY_PEERS Environment Variable Format

NDDS_DISCOVERY_PEERS can be specified via an environment variable of the same name, consisting of a sequence of peer descriptors separated by the comma (',') character.

Examples

Multicast (maximum participant ID is irrelevant)

- 239.255.0.1

Default maximum participant ID on localhost

- localhost

Default maximum participant ID on host 192.168.1.1 (IPv4)

- 192.168.1.1

Default maximum participant ID on host FAA0::0 (IPv6)

- FAA0::1

Default maximum participant ID on host himalaya accessed using the "udpv4" transport plugin(s) (IPv4)

- udpv4://himalaya

Default maximum participant ID on localhost using the "udpv4" transport plugin(s) registered at network address FAA0::0

- udpv4://FAA0::0#localhost

Default maximum participant ID on host 0/0/R (StarFabric)

- 0/0/R
- #0/0/R

Default maximum participant ID on host 0/0/R (StarFabric) using the "starfabric" (StarFabric) transport plugin(s)

- starfabric://0/0/R
- starfabric://#0/0/R

Default maximum participant ID on host 0/0/R (StarFabric) using the "starfabric" (StarFabric) transport plugin(s) registered at network address FAA0::0

- starfabric://FBB0::0#0/0/R

Default maximum participant ID on all unicast addresses accessed via the "starfabric" (StarFabric) transport plugin(s)

- starfabric://

Default maximum participant ID on all unicast addresses accessed via the "shmem" (shared memory) transport plugin(s) registered at network address FCC0::0

- shmem://FCC0::0

Default maximum participant ID on hosts himalaya and gangotri

- himalaya,gangotri

Maximum participant ID of 1 on hosts himalaya and gangotri

- 1@himalaya,1@gangotri

Combinations of above

- 239.255.0.1,localhost,192.168.1.1,0/0/R
- FAA0::1,FAA0::0#localhost,FBB0::0#0/0/R
- udpv4://himalaya,udpv4://FAA0::0#localhost,#0/0/R
- starfabric://0/0/R,starfabric://FBB0::0#0/0/R,shmem://
- starfabric://,shmem://FCC0::0,1@himalaya,1@gangotri

7.202.3 NDDS_DISCOVERY_PEERS File Format

NDDS_DISCOVERY_PEERS can be specified via a file of the same name in the program's current working directory. A NDDS_DISCOVERY_PEERS file would contain a sequence of peer descriptors separated by whitespace or the comma (',') character. The file may also contain comments starting with a semicolon (';') character till the end of the line.

Example:

```
;; NDDS_DISCOVERY_PEERS - Discovery Configuration File
;;
;;
;; NOTE:
;;   1. This file must be in the current working directory, i.e.
;;      in the folder from which the application is launched.
;;
;;   2. This file takes precedence over the environment variable NDDS_DISCOVERY_PEERS
;;
;; Multicast
```

```
239.255.0.1                 ; The default dds discovery multicast address
;; Unicast
localhost,192.168.1.1       ; A comma can be used a separator
FAA0::1  FAA0::0#localhost  ; Whitespace can be used as a separator
1@himalaya                  ; Maximum participant ID of 1 on 'himalaya'
1@gangotri
;; UDPv4
udpv4://himalaya            ; 'himalaya' via 'udpv4' transport plugin(s)
udpv4://FAA0::0#localhost   ; 'localhost' via 'updv4' transport
                            ;  plugin registered at network address FAA0::0
;; Shared Memory
shmem://                    ; All 'shmem' transport plugin(s)
builtin.shmem://            ; The builtin 'shmem' transport plugin
shmem://FCC0::0             ; Shared memory transport plugin registered
                            ;                at network address FCC0::0
;; StarFabric
0/0/R                       ; StarFabric node 0/0/R
starfabric://0/0/R          ; 0/0/R accessed via 'starfabric'
                            ;             transport plugin(s)
starfabric://FBB0::0#0/0/R  ; StarFabric transport plugin registered
                            ;             at network address FBB0::0
starfabric://               ; All 'starfabric' transport plugin(s)
```

7.202.4 NDDS_DISCOVERY_PEERS Precedence

If the current working directory from which the RTI Connext application is launched contains a file called NDDS_↩ DISCOVERY_PEERS, and an environment variable named NDDS_DISCOVERY_PEERS is also defined, the file takes precedence; the environment variable is ignored.

7.202.5 NDDS_DISCOVERY_PEERS Default Value

If NDDS_DISCOVERY_PEERS is not specified (either as a file in the current working directory, or as an environment variable), it implicitly defaults to the following.

```
;; Multicast (only on platforms which allow UDPv4 multicast out of the box)
;;
;; This allows any dds applications anywhere on the local network to
```

```
;; discover each other over UDPv4.
builtin.udpv4://239.255.0.1  ; dds's default discovery multicast address
                             ; This is also the default multicast receive address
;; Unicast - UDPv4 (on all platforms)
;;
;; This allows two dds applications using participant IDs up to the maximum
;; default participant ID on the local host and domain to discover each
;; other over UDP/IPv4.
builtin.udpv4://127.0.0.1
;; Unicast - Shared Memory (only on platforms that support shared memory)
;;
;; This allows two dds applications using participant IDs up to the maximum
;; default participant ID on the local host and domain to discover each
;; other over shared memory.
builtin.shmem://
```

7.202.6 Builtin Transport Class Names

The class names for the builtin transport plugins are:

- `shmem` - ::**Shared Memory Transport** (p. 456)
- `udpv4` - ::**UDPv4 Transport** (p. 462)
- `udpv6` - ::**UDPv6 Transport** (p. 479)

These may be used as the transport names in the **Locator Format** (p. 663).

7.202.7 NDDS_DISCOVERY_PEERS and Local Host Communication

Suppose you want to communicate with other RTI Connext applications on the same host and you are setting NDDS↩ _DISCOVERY_PEERS explicitly (generally in order to use unicast discovery with applications on other hosts).

If the local host platform does not support the shared memory transport, then you can include the name of the local host in the NDDS_DISCOVERY_PEERS list.

If the local host platform supports the shared memory transport, then you can do one of the following:

- Include "shmem://" in the NDDS_DISCOVERY_PEERS list. This will cause shared memory to be used for discovery and data traffic for applications on the same host.

or:

- Include the name of the local host in the NDDS_DISCOVERY_PEERS list and disable the shared memory transport in the **DDS_TransportBuiltinQosPolicy** (p. 1408) of the **DDSDomainParticipant** (p. 1619). This will cause UDP loopback to be used for discovery and data traffic for applications on the same host.

(To check if your platform supports shared memory, see the `Platform Notes`.)

See also

DDS_DiscoveryQosPolicy::multicast_receive_addresses (p. 962)

DDS_DiscoveryQosPolicy::initial_peers (p. 962)

DDSDomainParticipant::add_peer() (p. 1678)

DDS_PARTICIPANT_QOS_DEFAULT (p. 50)

DDSDomainParticipantFactory::get_default_participant_qos() (p. 1700)

Transport Aliases (p. 173)

Transport Network Address (p. 176)

NDDSTransportSupport::register_transport() (p. 2104)

7.203 Entity Support

DDSEntity (p. 1731), **DDSListener** (p. 1794) and related items.

Classes

- class **DDSListener**

  *<<**interface**>> (p. 433) Abstract base class for all Listener interfaces.*
- class **DDSEntity**

  *<<**interface**>> (p. 433) Abstract base class for all the DDS objects that support QoS policies, a listener, and a status condition.*
- class **DDSDomainEntity**

  *<<**interface**>> (p. 433) Abstract base class for all DDS entities except for the **DDSDomainParticipant** (p. 1619).*

7.203.1 Detailed Description

DDSEntity (p. 1731), **DDSListener** (p. 1794) and related items.

DDSEntity (p. 1731) subtypes are created and destroyed by factory objects. With the exception of **DDSDomain↩ Participant** (p. 1619), whose factory is **DDSDomainParticipantFactory** (p. 1694), all **DDSEntity** (p. 1731) factory objects are themselves **DDSEntity** (p. 1731) subtypes as well.

Important: all **DDSEntity** (p. 1731) delete operations are inherently thread-unsafe. The user must take extreme care that a given **DDSEntity** (p. 1731) is not destroyed in one thread while being used concurrently (including being deleted concurrently) in another thread. An operation's effect in the presence of the concurrent deletion of the operation's target **DDSEntity** (p. 1731) is undefined.

7.204 Conditions and WaitSets

DDSCondition (p. 1544) and **DDSWaitSet** (p. 1897) and related items.

Modules

- **AsyncWaitSet**

  *<<**extension**>> (p. 432) A specialization of **DDSWaitSet** (p. 1897) that provides a mechanism to perform the wait asynchronously and uses a thread pool to dispatch the attached active **DDSCondition** (p. 1544).*

Classes

- struct **DDS_WaitSetProperty_t**

  *<<**extension**>> (p. 432) Specifies the **DDSWaitSet** (p. 1897) behavior for multiple trigger events.*
- struct **DDSConditionSeq**

  *Instantiates **FooSeq** (p. 1965) < **DDSCondition** (p. 1544) >*
- class **DDSConditionHandler**

  *<<**extension**>> (p. 432) <<**interface**>> (p. 433) Handler called by the **DDSCondition::dispatch** (p. 1546).*
- class **DDSCondition**

  *<<**interface**>> (p. 433) Root class for all the conditions that may be attached to a **DDSWaitSet** (p. 1897).*
- class **DDSGuardCondition**

  *<<**interface**>> (p. 433) A specific **DDSCondition** (p. 1544) whose `trigger_value` is completely under the control of the application.*
- class **DDSStatusCondition**

  *<<**interface**>> (p. 433) A specific **DDSCondition** (p. 1544) that is associated with each **DDSEntity** (p. 1731).*
- class **DDSWaitSet**

  *<<**interface**>> (p. 433) Allows an application to wait until one or more of the attached **DDSCondition** (p. 1544) objects has a `trigger_value` of **DDS_BOOLEAN_TRUE** (p. 513) or else until the timeout expires.*

Macros

- #define **DDS_WaitSetProperty_t_INITIALIZER**

  *<<**extension**>> (p. 432) Initializer for new property instances.*

7.204.1 Detailed Description

DDSCondition (p. 1544) and **DDSWaitSet** (p. 1897) and related items.

7.204.2 Macro Definition Documentation

7.204.2.1 DDS_WaitSetProperty_t_INITIALIZER

```
#define DDS_WaitSetProperty_t_INITIALIZER
```

Value:

```
   {  \
   1, DDS_DURATION_INFINITE_VALUE \
}
```

<<***extension***>> *(*p. *432)* Initializer for new property instances.

Default property specifies `max_event_count` = 1 and `max_event_delay` = `DDS_DURATION_INFINITE`

7.205 Cookie

<<***extension***>> *(*p. *432)* Unique identifier for a written data sample

Classes

- struct **DDS_Cookie_t**

  <<***extension***>> *(*p. *432) Sequence of bytes.*
- struct **DDS_CookieSeq**

  Declares IDL `sequence` < ***DDS_Cookie_t*** *(*p. *830)* > .

Functions

- void * **DDS_Cookie_t::to_pointer** () const

  Returns the pointer previously stored in the cookie.

7.205.1 Detailed Description

<<***extension***>> *(*p. *432)* Unique identifier for a written data sample

7.205.2 Function Documentation

7.205.2.1 to_pointer()

```
void * DDS_Cookie_t::to_pointer ( ) const
```

Returns the pointer previously stored in the cookie.

Precondition

The cookie's value was filled with a pointer; otherwise, the value returned may not be a valid memory address.

Returns

The pointer stored in the cookie.

7.206 Sample Flags

<<***extension***>> *(p. 432)* Flags for samples.

Typedefs

- typedef enum **DDS_SampleFlagBits** **DDS_SampleFlagBits**
  Type to identify the sample flags reserved by RTI.
- typedef **DDS_Long** **DDS_SampleFlag**
  A set of flags that can be associated with a sample.

Enumerations

- enum **DDS_SampleFlagBits** {
  DDS_REDELIVERED_SAMPLE ,
  DDS_INTERMEDIATE_REPLY_SEQUENCE_SAMPLE ,
  DDS_REPLICATE_SAMPLE ,
  DDS_LAST_SHARED_READER_QUEUE_SAMPLE ,
  DDS_INTERMEDIATE_TOPIC_QUERY_SAMPLE ,
  DDS_INCOMPLETE_SNAPSHOT_TOPIC_QUERY ,
  DDS_WRITER_REMOVED_BATCH_SAMPLE = PRES_WRITER_REMOVED_BATCH_SAMPLE ,
  DDS_DISCOVERY_SERVICE_SAMPLE = PRES_DISCOVERY_SERVICE_SAMPLE }
  Type to identify the sample flags reserved by RTI.

7.206.1 Detailed Description

<<***extension***>> *(p. 432)* Flags for samples.

7.206.2 Typedef Documentation

7.206.2.1 DDS_SampleFlagBits

`typedef enum` **`DDS_SampleFlagBits DDS_SampleFlagBits`**

Type to identify the sample flags reserved by RTI.

7.206.2.2 DDS_SampleFlag

`typedef` **`DDS_Long DDS_SampleFlag`**

A set of flags that can be associated with a sample.

- Least-significant bits [0-7] are reserved by RTI
- Least-significant bits [8-15] are application specific
- Least-significant bits [16-31] are invalid and cannot be used

7.206.3 Enumeration Type Documentation

7.206.3.1 DDS_SampleFlagBits

`enum` **`DDS_SampleFlagBits`**

Type to identify the sample flags reserved by RTI.

Enumerator

DDS_REDELIVERED_SAMPLE	Indicates that a sample has been redelivered by RTI Queuing Service.
DDS_INTERMEDIATE_REPLY_SEQUENCE_SAMPLE	Indicates that a response sample is not the last response sample for a given request. This bit is usually set by Connext Repliers sending multiple responses for a request.
DDS_REPLICATE_SAMPLE	Indicates if a sample must be broadcast by one RTI Queuing Service replica to other replicas.

Enumerator

DDS_LAST_SHARED_READER_QUEUE_SAMPLE	Indicates that a sample is the last sample in a SharedReaderQueue for a QueueConsumer DataReader.
DDS_INTERMEDIATE_TOPIC_QUERY_SAMPLE	Indicates that a sample for a TopicQuery will be followed by more samples. This flag only applies to samples that have been published as a response to a **DDSTopicQuery** (p. 1895). When this bit is not set and **DDS_SampleInfo::topic_query_guid** (p. 1354) is different from **DDS_GUID_UNKNOWN** (p. 527), this sample is the last sample for that TopicQuery coming from the DataWriter identified by **DDS_SampleInfo::source_guid** (p. 1353).
DDS_INCOMPLETE_SNAPSHOT_TOPIC_QUERY	Indicates that some of the samples that were published as part of a snapshot TopicQuery were lost. This flag is set in the last sample of the snapshot.
DDS_WRITER_REMOVED_BATCH_SAMPLE	This flag will be set if the sample was accepted into the DataReader queue even though it was marked by the DataWriter as removed. A sample can be marked as removed by the DataWriter in a batch when it is replaced due to the **DDS_KEEP_LAST_HISTORY_QOS** (p. 604) **DDS_HistoryQosPolicy** (p. 1162) QoS or because the duration in **DDS_LifespanQosPolicy** (p. 1175) was reached. If the DataReader sets the property "dds.data_reader.↩ accept_writer_removed_batch_samples" to true, the removed sample will be accepted into the DataReader queue and this flag will be set.
DDS_DISCOVERY_SERVICE_SAMPLE	This flag will be set if the sample was sent by Cloud Discovery Service. The samples sent by Cloud Discovery Service are participant announcement samples on the ParticipantBuiltinTopic.

7.207 WriteParams

*<<**extension**>> (p. 432)*

Classes

- struct **DDS_SampleIdentity_t**

  Type definition for a Sample Identity.
- struct **DDS_AckResponseData_t**

  Data payload of an application-level acknowledgment.
- struct **DDS_WriteParams_t**

  *<<**extension**>> (p. 432) Input parameters for writing with **FooDataWriter::write_w_params** (p. 1956), **FooData↩ Writer::dispose_w_params** (p. 1959), **FooDataWriter::register_instance_w_params** (p. 1948), **FooDataWriter↩ ::unregister_instance_w_params** (p. 1951)*

Functions

- **DDS_Boolean DDS_SampleIdentity_equals** (const struct **DDS_SampleIdentity_t** *self, const struct **DDS_SampleIdentity_t** *other)

  Compares this sample identity with another sample identity for equality.
- void **DDS_WriteParams_reset** (struct **DDS_WriteParams_t** *self)

  Resets all the fields to their default values.

Variables

- struct **DDS_GUID_t DDS_SampleIdentity_t::writer_guid**

  16-byte identifier identifying the virtual GUID.
- struct **DDS_SequenceNumber_t DDS_SampleIdentity_t::sequence_number**

  monotonically increasing 64-bit integer that identifies the sample in the data source.
- const struct **DDS_SampleIdentity_t DDS_AUTO_SAMPLE_IDENTITY**

  The AUTO sample identity.
- const struct **DDS_SampleIdentity_t DDS_UNKNOWN_SAMPLE_IDENTITY**

  An invalid or unknown sample identity.
- const struct **DDS_WriteParams_t DDS_WRITEPARAMS_DEFAULT**

  *Initializer for **DDS_WriteParams_t** (p. 1519).*

7.207.1 Detailed Description

*<<**extension**>> (p. 432)*

7.207.2 Function Documentation

7.207.2.1 DDS_SampleIdentity_equals()

```
DDS_Boolean DDS_SampleIdentity_equals (
            const struct  DDS_SampleIdentity_t * self,
            const struct  DDS_SampleIdentity_t * other )
```

Compares this sample identity with another sample identity for equality.

Parameters

self	*<<**in**>> (p. 433)* This sample identity.
other	*<<**in**>> (p. 433)* The other sample identity to be compared with this sample identity.

Returns

DDS_BOOLEAN_TRUE (p. 513) if the two sample identities have equal values, or **DDS_BOOLEAN_FALSE** (p. 513) otherwise.

7.207.2.2 DDS_WriteParams_reset()

```
void DDS_WriteParams_reset (
            struct  DDS_WriteParams_t * self )
```

Resets all the fields to their default values.

This operation is useful to reset all the fields to their automatic value when **DDS_WriteParams_t::replace_auto** (p. 1520) is enabled and the same params instance is used in multiple calls to **FooDataWriter::write_w_params** (p. 1956)

7.207.3 Variable Documentation

7.207.3.1 writer_guid

```
struct  DDS_GUID_t DDS_SampleIdentity_t::writer_guid
```

16-byte identifier identifying the virtual GUID.

7.207.3.2 sequence_number

```
struct  DDS_SequenceNumber_t DDS_SampleIdentity_t::sequence_number
```

monotonically increasing 64-bit integer that identifies the sample in the data source.

7.207.3.3 DDS_AUTO_SAMPLE_IDENTITY

```
const struct  DDS_SampleIdentity_t DDS_AUTO_SAMPLE_IDENTITY  [extern]
```

The AUTO sample identity.

Special **DDS_AUTO_SAMPLE_IDENTITY** (p. 676) value {**DDS_GUID_AUTO** (p. 527), **DDS_AUTO_SEQUENCE_↩ NUMBER** (p. 529)}

7.207.3.4 DDS_UNKNOWN_SAMPLE_IDENTITY

```
const struct  DDS_SampleIdentity_t DDS_UNKNOWN_SAMPLE_IDENTITY  [extern]
```

An invalid or unknown sample identity.

Special **DDS_UNKNOWN_SAMPLE_IDENTITY** (p. 676) value {**DDS_GUID_UNKNOWN** (p. 527), **DDS_SEQUENCE_NUMBER_UNKNOWN** (p. 528)}

7.207.3.5 DDS_WRITEPARAMS_DEFAULT

```
const struct  DDS_WriteParams_t DDS_WRITEPARAMS_DEFAULT  [extern]
```

Initializer for **DDS_WriteParams_t** (p. 1519).

7.208 Heap Support in C

<<***extension***>> *(*p. *432)* Heap allocation and free routines in C

Functions

- void * **DDS_Heap_calloc** (size_t numElem, size_t size)

  Peforms the logical equivalent of calloc().
- void * **DDS_Heap_malloc** (size_t size)

  Peforms the logical equivalent of malloc().
- void **DDS_Heap_free** (void *ptr)

  Performs the logical equivalent of free().

7.208.1 Detailed Description

<<***extension***>> *(*p. *432)* Heap allocation and free routines in C

The methods in this class ensure consistent cross-platform implementations for memory allocation (**DDS_Heap_malloc()** (p. 678) and **DDS_Heap_calloc()** (p. 678)) and deletion (**DDS_Heap_free()** (p. 678)).

Applications need to use these routines to reserve memory that the middleware will release or to free memory that the middleware has allocated.

For example, to allocate an optional member in a sample use **DDS_Heap_malloc()** (p. 678)/ **DDS_Heap_calloc()** (p. 678); to release an optional member previously reserved by **FooTypeSupport::create_data()** (p. 1983), use **DDS_Heap_free()** (p. 678).

7.208.2 Function Documentation

7.208.2.1 DDS_Heap_calloc()

```
void * DDS_Heap_calloc (
            size_t numElem,
            size_t size )
```

Peforms the logical equivalent of calloc().

Allocates unused space for an array with `numElem` elements each of whose size in bytes is `size`. The memory will be initialized to zeros.

Parameters

numElem	<<***in***>> *(*p. *433)* The number of elements in the array to be allocated
size	<<***in***>> *(*p. *433)* The size in bytes of each element in the array

Returns

If insufficient memory is available, this method will return NULL. Otherwise, upon success it returns a pointer to the allocated space.

7.208.2.2 DDS_Heap_malloc()

```
void * DDS_Heap_malloc (
            size_t size )
```

Peforms the logical equivalent of malloc().

Allocates unused space for an object of the specified `size`. The memory will be initialized to zeros.

Parameters

size	<<***in***>> *(*p. *433)* The size in bytes of the object to be allocated

Returns

If insufficient memory is available, this method will return NULL. Otherwise, upon success it returns a pointer to the allocated space.

7.208.2.3 DDS_Heap_free()

```
void DDS_Heap_free (
             void * ptr )
```

Performs the logical equivalent of free().

Deallocates the space pointed to by `ptr` and made available for further allocation.

Parameters

ptr	<<***in***>> *(*p. *433)* A pointer to the space to be deallocated

7.209 Builtin Qos Profiles

<<***extension***>> *(*p. *432)* QoS libraries, profiles, and snippets that are automatically built into RTI Connext.

Variables

- const char ∗const **DDS_BUILTIN_QOS_LIB**

  A library of non-experimental QoS profiles.
- const char ∗const **DDS_PROFILE_BASELINE_ROOT**

  The root baseline QoS values from which all other Baseline.X.X.X profiles inherit.
- const char ∗const **DDS_PROFILE_BASELINE**

  The most up-to-date QoS default values.
- const char ∗const **DDS_PROFILE_BASELINE_5_0_0**

  The QoS default values for version 5.0.0.
- const char ∗const **DDS_PROFILE_BASELINE_5_1_0**

  The QoS default values for version 5.1.0.
- const char ∗const **DDS_PROFILE_BASELINE_5_2_0**

  The QoS default values for version 5.2.0.
- const char ∗const **DDS_PROFILE_BASELINE_5_3_0**

  The QoS default values for version 5.3.0.
- const char ∗const **DDS_PROFILE_BASELINE_6_0_0**

  The QoS default values for version 6.0.0.
- const char ∗const **DDS_PROFILE_BASELINE_6_1_0**

  The QoS default values for version 6.1.0.
- const char ∗const **DDS_PROFILE_BASELINE_7_0_0**

  The QoS default values for version 7.0.0.
- const char ∗const **DDS_PROFILE_BASELINE_7_1_0**

  The QoS default values for version 7.1.0.
- const char ∗const **DDS_PROFILE_GENERIC_COMMON**

  A common Participant base profile.
- const char ∗const **DDS_PROFILE_GENERIC_MONITORING_COMMON**

Enables RTI Monitoring Library.

- const char *const **DDS_PROFILE_GENERIC_CONNEXT_MICRO_COMPATIBILITY**

  Sets the values necessary to communicate with RTI Connext Micro.
- const char *const **DDS_PROFILE_GENERIC_CONNEXT_MICRO_COMPATIBILITY_2_4_9**

  Sets the values necessary to communicate with RTI Connext Micro versions 2.4.4 through at least 2.4.9.
- const char *const **DDS_PROFILE_GENERIC_CONNEXT_MICRO_COMPATIBILITY_2_4_3**

  Sets the values necessary to communicate with RTI Connext Micro versions 2.4.3 and ealier.
- const char *const **DDS_PROFILE_GENERIC_OTHER_DDS_VENDOR_COMPATIBILITY**

  Sets the values necessary to interoperate with other DDS vendors.
- const char *const **DDS_PROFILE_GENERIC_510_TRANSPORT_COMPATIBILITY**

  Sets the values necessary to interoperate with RTI Connext 5.1.0 using the UDPv6 and/or SHMEM transports.
- const char *const **DDS_BUILTIN_QOS_LIB_EXP**

  A library of experimental QoS profiles.
- const char *const **DDS_PROFILE_GENERIC_STRICT_RELIABLE**

  Enables strict reliability.
- const char *const **DDS_PROFILE_GENERIC_KEEP_LAST_RELIABLE**

  Enables keep-last reliability.
- const char *const **DDS_PROFILE_GENERIC_BEST_EFFORT**

  Enables best-effort reliability kind.
- const char *const **DDS_PROFILE_GENERIC_STRICT_RELIABLE_HIGH_THROUGHPUT**

  A profile that can be used to achieve high throughput.
- const char *const **DDS_PROFILE_GENERIC_STRICT_RELIABLE_LOW_LATENCY**

  A profile that can be used to achieve low latency.
- const char *const **DDS_PROFILE_GENERIC_PARTICIPANT_LARGE_DATA**

  A common Participant base profile to facilitate sending large data.
- const char *const **DDS_PROFILE_GENERIC_PARTICIPANT_LARGE_DATA_MONITORING**

  Configures Participants for large data and monitoring.
- const char *const **DDS_PROFILE_GENERIC_STRICT_RELIABLE_LARGE_DATA**

  Configures endpoints for sending large data with strict reliability.
- const char *const **DDS_PROFILE_GENERIC_KEEP_LAST_RELIABLE_LARGE_DATA**

  Configures endpoints for sending large data with keep-last reliability.
- const char *const **DDS_PROFILE_GENERIC_STRICT_RELIABLE_LARGE_DATA_FAST_FLOW**

  Configures strictly reliable communication for large data with a fast flow controller.
- const char *const **DDS_PROFILE_GENERIC_STRICT_RELIABLE_LARGE_DATA_MEDIUM_FLOW**

  Configures strictly reliable communication for large data with a medium flow controller.
- const char *const **DDS_PROFILE_GENERIC_STRICT_RELIABLE_LARGE_DATA_SLOW_FLOW**

  Configures strictly reliable communication for large data with a slow flow controller.
- const char *const **DDS_PROFILE_GENERIC_KEEP_LAST_RELIABLE_LARGE_DATA_FAST_FLOW**

  Configures keep-last reliable communication for large data with a fast flow controller.
- const char *const **DDS_PROFILE_GENERIC_KEEP_LAST_RELIABLE_LARGE_DATA_MEDIUM_FLOW**

  Configures keep-last reliable communication for large data with a medium flow controller.
- const char *const **DDS_PROFILE_GENERIC_KEEP_LAST_RELIABLE_LARGE_DATA_SLOW_FLOW**

  Configures keep-last reliable communication for large data with a slow flow controller.
- const char *const **DDS_PROFILE_GENERIC_KEEP_LAST_RELIABLE_TRANSIENT_LOCAL**

  Persists the samples of a DataWriter as long as the entity exists.
- const char *const **DDS_PROFILE_GENERIC_KEEP_LAST_RELIABLE_TRANSIENT**

  Persists samples using RTI Persistence Service.

- const char ∗const **DDS_PROFILE_GENERIC_KEEP_LAST_RELIABLE_PERSISTENT**

  Persists samples in permanent storage, like a disk, using RTI Persistence Service.
- const char ∗const **DDS_PROFILE_GENERIC_AUTO_TUNING**

  Enables the Turbo Mode batching and Auto Throttle experimental features.
- const char ∗const **DDS_PROFILE_GENERIC_MINIMAL_MEMORY_FOOTPRINT**

  Uses a set of QoS which reduces the memory footprint of the application.
- const char ∗const **DDS_PROFILE_GENERIC_SECURITY**

  Loads the DDS Secure builtin plugins.
- const char ∗const **DDS_PROFILE_GENERIC_MONITORING2**

  The default QoS profile that the DDS entities created by the RTI Monitoring Library 2.0 use.
- const char ∗const **DDS_PROFILE_GENERIC_MONITORING2_WAN**

  The QoS profile that the DDS entities created by the RTI Monitoring Library 2.0 use to communicate over WAN with a Collector Service.
- const char ∗const **DDS_PROFILE_PATTERN_RPC**

  This profile contains the default RPC and Request-Reply QoS settings.
- const char ∗const **DDS_PROFILE_PATTERN_PERIODIC_DATA**

  Used for applications that expect periodic data.
- const char ∗const **DDS_PROFILE_PATTERN_STREAMING**

  Used for applications that stream data.
- const char ∗const **DDS_PROFILE_PATTERN_RELIABLE_STREAMING**

  Used for applications that stream data and require reliable communication.
- const char ∗const **DDS_PROFILE_PATTERN_EVENT**

  Used for applications that handle events.
- const char ∗const **DDS_PROFILE_PATTERN_ALARM_EVENT**

  Used for applications that handle alarm events.
- const char ∗const **DDS_PROFILE_PATTERN_STATUS**

  Used for applications whose samples represent statuses.
- const char ∗const **DDS_PROFILE_PATTERN_ALARM_STATUS**

  Used for applications in which samples represent alarm statuses.
- const char ∗const **DDS_PROFILE_PATTERN_LAST_VALUE_CACHE**

  Used for applications that only need the last published value.
- const char ∗const **DDS_BUILTIN_QOS_SNIPPET_LIB**

  A library of QoS Snippets.
- const char ∗const **DDS_SNIPPET_OPTIMIZATION_RELIABILITY_PROTOCOL_COMMON**

  QoS Snippet that configures the reliability protocol with a common configuration.
- const char ∗const **DDS_SNIPPET_OPTIMIZATION_RELIABILITY_PROTOCOL_KEEP_ALL**

  QoS Snippet that configures the reliability protocol for KEEP_ALL.
- const char ∗const **DDS_SNIPPET_OPTIMIZATION_RELIABILITY_PROTOCOL_KEEP_LAST**

  QoS Snippet that configures the reliability protocol for KEEP_LAST.
- const char ∗const **DDS_SNIPPET_OPTIMIZATION_RELIABILITY_PROTOCOL_HIGH_RATE**

  QoS Snippet that configures the reliability protocol for sending data at a high rate.
- const char ∗const **DDS_SNIPPET_OPTIMIZATION_RELIABILITY_PROTOCOL_LOW_LATENCY**

  QoS Snippet that configures the reliability protocol for sending data at low latency.
- const char ∗const **DDS_SNIPPET_OPTIMIZATION_RELIABILITY_PROTOCOL_LARGE_DATA**

  QoS Snippet that configures the reliability protocol for large data.
- const char ∗const **DDS_SNIPPET_OPTIMIZATION_DATACACHE_LARGE_DATA_DYNAMICMEMALLOC**

  Configures the DataWriter and DataReader caches to use dynamic memory allocation for Large Data samples.

- const char *const **DDS_SNIPPET_OPTIMIZATION_DISCOVERY_COMMON**

  QoS Snippet that optimizes discovery with a common configuration.
- const char *const **DDS_SNIPPET_OPTIMIZATION_DISCOVERY_PARTICIPANT_COMPACT**

  QoS Snippet that optimizes the Participant QoS to send less discovery information.
- const char *const **DDS_SNIPPET_OPTIMIZATION_DISCOVERY_ENDPOINT_FAST**

  QoS Snippet that optimizes the Endpoint Discovery to be faster.
- const char *const **DDS_SNIPPET_OPTIMIZATION_TRANSPORT_LARGE_BUFFERS**

  QoS Snippet that increases the Participant default buffer that shmem and udpv4 use.
- const char *const **DDS_SNIPPET_QOS_POLICY_RELIABILITY_RELIABLE**

  QoS Snippet that sets RELIABILITY QoS to RELIABLE.
- const char *const **DDS_SNIPPET_QOS_POLICY_RELIABILITY_BEST_EFFORT**

  QoS Snippet that sets RELIABILITY QoS to BEST_EFFORT.
- const char *const **DDS_SNIPPET_QOS_POLICY_HISTORY_KEEP_LAST_1**

  QoS Snippet that sets HISTORY QoS to KEEP_LAST kind with depth 1.
- const char *const **DDS_SNIPPET_QOS_POLICY_HISTORY_KEEP_ALL**

  QoS Snippet that sets HISTORY QosPolicy to KEEP_ALL kind.
- const char *const **DDS_SNIPPET_QOS_POLICY_PUBLISH_MODE_ASYNCHRONOUS**

  QoS Snippet that sets PUBLISH_MODE QosPolicy to ASYNCHRONOUS kind.
- const char *const **DDS_SNIPPET_QOS_POLICY_DURABILITY_TRANSIENT_LOCAL**

  QoS Snippet that sets DURABILITY QosPolicy to TRANSIENT_LOCAL kind.
- const char *const **DDS_SNIPPET_QOS_POLICY_DURABILITY_TRANSIENT**

  QoS Snippet that sets DURABILITY QosPolicy to TRANSIENT kind.
- const char *const **DDS_SNIPPET_QOS_POLICY_DURABILITY_PERSISTENT**

  QoS Snippet that sets DURABILITY QosPolicy to PERSISTENT kind.
- const char *const **DDS_SNIPPET_QOS_POLICY_BATCHING_ENABLE**

  QoS Snippet that sets BATCH QosPolicy to true.
- const char *const **DDS_SNIPPET_FEATURE_FLOW_CONTROLLER_838MBPS**

  QoS Snippet that configures and set a FlowController of 838 Mbps.
- const char *const **DDS_SNIPPET_FEATURE_FLOW_CONTROLLER_209MBPS**

  QoS Snippet that configures and sets a FlowController of 209 Mbps.
- const char *const **DDS_SNIPPET_FEATURE_FLOW_CONTROLLER_52MBPS**

  QoS Snippet that configures and sets a FlowController of 52 Mbps.
- const char *const **DDS_SNIPPET_FEATURE_AUTO_TUNING_ENABLE**

  QoS Snippet that enables auto_throttle and turbo_mode to true.
- const char *const **DDS_SNIPPET_FEATURE_MONITORING_ENABLE**

  QoS Snippet that enables the use of the RTI Monitoring Library.
- const char *const **DDS_SNIPPET_FEATURE_MONITORING2_ENABLE**

  QoS Snippet that enables the use of the RTI Monitoring Library 2.0.
- const char *const **DDS_SNIPPET_FEATURE_SECURITY_ENABLE**

  QoS Snippet that enables security using the Builtin Security Plugins.
- const char *const **DDS_SNIPPET_FEATURE_TOPIC_QUERY_ENABLE**

  QoS Snippet that enables Topic Query.
- const char *const **DDS_SNIPPET_FEATURE_TYPE_SUPPORT_V1_ENABLE**

  QoS Snippet that enables TypeObject V1 propagation.
- const char *const **DDS_SNIPPET_FEATURE_TYPE_SUPPORT_V1_DISABLE**

  QoS Snippet that disables TypeObject V1 propagation.
- const char *const **DDS_SNIPPET_FEATURE_TYPE_SUPPORT_V2_ENABLE**

QoS Snippet that enables TypeObject V2 propagation.

- const char *const **DDS_SNIPPET_FEATURE_TYPE_SUPPORT_V2_DISABLE**

  QoS Snippet that disables TypeObject V2 propagation.
- const char *const **DDS_SNIPPET_TRANSPORT_TCP_LAN_CLIENT**

  QoS Snippet that configures a TCP LAN Client over DDS.
- const char *const **DDS_SNIPPET_TRANSPORT_TCP_WAN_SYMMETRIC_CLIENT**

  QoS Snippet that configures a symmetric WAN TCP Client over DDS.
- const char *const **DDS_SNIPPET_TRANSPORT_TCP_WAN_ASYMMETRIC_SERVER**

  QoS Snippet that an asymmetric WAN TCP Server over DDS.
- const char *const **DDS_SNIPPET_TRANSPORT_TCP_WAN_ASYMMETRIC_CLIENT**

  QoS Snippet that configures an asymmetric WAN TCP Client over DDS.
- const char *const **DDS_SNIPPET_TRANSPORT_UDP_AVOID_IP_FRAGMENTATION**

  QoS Snippet that configures RTI Connext and the UDP built-in transports (UDPv4, UDPv6, UDPv4_WAN) to avoid IP fragmentation.
- const char *const **DDS_SNIPPET_TRANSPORT_UDP_WAN**

  QoS Snippet that enables the RTI Real-Time WAN Transport (UDPv4_WAN).
- const char *const **DDS_SNIPPET_COMPATIBILITY_CONNEXT_MICRO_VERSION_2_4_3**

  QoS Snippet that configures sets LIVELINESS QoS to be compatible with RTI Connext Micro 2.4.3.
- const char *const **DDS_SNIPPET_COMPATIBILITY_OTHER_DDS_VENDORS_ENABLE**

  QoS Snippet that configures RTI Connext to interoperate with other DDS vendors.
- const char *const **DDS_SNIPPET_5_1_0_TRANSPORT_ENABLE**

  QoS Snippet that configures RTI Connext to interoperate with RTI Connext 5.1.0 and below for UDPv6 and SHMEM transports.

7.209.1 Detailed Description

<<***extension***>> *(*p. *432)* QoS libraries, profiles, and snippets that are automatically built into RTI Connext.

The built-in profiles can be accessed in QoS XML configuration files and by using any of the APIs that accept library and profiles names by using the constants or string versions as documented on this page.

The built-in profiles are provided as a way to quickly and easily configure RTI Connext applications with a set of QoS values aimed at achieving a specific behavior.

There are three built-in QoS libraries:

- BuiltinQosLib: A library containing built-in QoS Profiles.

- BuiltinQosLibExp: A library containing experimental QoS Profiles. Experimental QoS Profiles are new QoS Profiles that have been tested internally but have not gone through an extensive validation period. Therefore, some of the settings may change in future releases based on customer and internal feedback. After validation, experimental QoS Profiles will be moved into the non-experimental library.

- BuiltinQosSnippetLib: A library containing QoS Snippets that are ready to use as elements for the QoS Profile composition pattern. For further information about this pattern visit the following article: `https://community.rti.com/best-practices/qos-profile-inheritance-and-composition-guidance`

There are three types of profiles:

- Baseline.X.X.X: These profiles represent the QoS defaults for /ndds version X.X.X. To access the defaults for the latest RTI Connext version, use the BuiltinQosLib::Baseline profile.
- Generic.X: These profiles allow you to easily configure different features and communication use-cases with RTI Connext. For example, there is a Generic.StrictReliable profile for use when your application has a requirement for no data loss.
- Pattern.X: These profiles inherit from the Generic.X profiles and allow you to configure various domain-specific communication use cases. For example, there is a Pattern.Alarm profile that can be used to manage the generation and consumption of alarm events.

There are several types of QoS Snippets. These are the current QoS Snippets available:

- Optimization.X: these QoS Snippets optimize one or more parameters related to the X QoS Policy or a specific use-case.
- QosPolicy.X.Y: these QoS Snippets set a specific QoS Policy X to the value Y.
- Feature.X: these QoS Snippets set all the needed QoS values to enable/modify a specific feature.
- Transport.X: these QoS Snippets set a specific transport defined by X. This transport may have specific scenarios that are also specified in the name.
- Compatibility.X: these QoS Snippets change the specific QoS policies to ensure compatibility with specific products or versions specified by X.

These QoS Profiles can be used as base profiles in XML configuration, then QoS Snippets can modify specific aspects of this base QoS Profile, and finally files and values can be modified to fit a specific system's needs. For example:

```
<qos_profile name="MyProfile" base_name="BuiltinQosLib::Generic.Common">

   <domain_participant_qos base_name="BuiltinQosLib::Generic.Monitoring.Common">

     <base_name>

       <element>BuiltinQosSnippetLib::Feature.Monitoring.Enable</element>

     </base_name>

     <!-- Override and add values -->

   </domain_participant_qos>

</qos_profile>
```

The QoS Profiles can also be used in any APIs that call for a QoS library and profile name, for example, the create_*↩ _with_profile() APIs. To create a **DDSDataWriter** (p. 1589) configured to send large data asynchronously:

```
writer = DDS_DomainParticipant_create_datawriter_with_profile(

                 participant, topic,

                 ::DDS_BUILTIN_QOS_LIB,

                 ::DDS_PROFILE_GENERIC_STRICT_RELIABLE_LARGE_DATA_FAST_FLOW,

                 NULL, ::DDS_STATUS_MASK_NONE);
```

All the built-in QoS Profiles and QoS Snippets are documented in the BaselineRoot.documentationONLY.xml and BuiltinProfiles.documentationONLY.xml files that are included in the NDDSHOME/xml directory of the RTI Connext installation.

- BaselineRoot.documentationONLY.xml contains the root baseline QoS profile that corresponds to the default values of RTI Connext 5.0.0.
- BuiltinProfiles.documentationONLY.xml contains the rest of the built-in Qos Profiles and QoS Snippets.

7.209.2 Variable Documentation

7.209.2.1 DDS_BUILTIN_QOS_LIB

```
const char* const DDS_BUILTIN_QOS_LIB  [extern]
```

A library of non-experimental QoS profiles.

String-version: "BuiltinQosLib"

7.209.2.2 DDS_PROFILE_BASELINE_ROOT

```
const char* const DDS_PROFILE_BASELINE_ROOT  [extern]
```

The root baseline QoS values from which all other Baseline.X.X.X profiles inherit.

This profile contains the root baseline QoS values from which all other Baseline.X.X.X profiles inherit. These values correspond to the default values for RTI Connext 5.0.0.

In Library: **DDS_BUILTIN_QOS_LIB** (p. 685)

String-version: "Baseline.Root"

7.209.2.3 DDS_PROFILE_BASELINE

```
const char* const DDS_PROFILE_BASELINE  [extern]
```

The most up-to-date QoS default values.

You can use this profile if you want your application to pick up and use any new QoS default settings each time a new RTI Connext version is released – without changing your application code.

In Library: **DDS_BUILTIN_QOS_LIB** (p. 685)

String-version: "Baseline"

7.209.2.4 DDS_PROFILE_BASELINE_5_0_0

```
const char* const DDS_PROFILE_BASELINE_5_0_0  [extern]
```

The QoS default values for version 5.0.0.

In Library: **DDS_BUILTIN_QOS_LIB** (p. 685)

String-version: "Baseline.5.0.0"

7.209.2.5 DDS_PROFILE_BASELINE_5_1_0

```
const char* const DDS_PROFILE_BASELINE_5_1_0  [extern]
```

The QoS default values for version 5.1.0.

In Library: **DDS_BUILTIN_QOS_LIB** (p. 685)

String-version: "Baseline.5.1.0"

7.209.2.6 DDS_PROFILE_BASELINE_5_2_0

```
const char* const DDS_PROFILE_BASELINE_5_2_0  [extern]
```

The QoS default values for version 5.2.0.

In Library: **DDS_BUILTIN_QOS_LIB** (p. 685)

String-version: "Baseline.5.2.0"

7.209.2.7 DDS_PROFILE_BASELINE_5_3_0

```
const char* const DDS_PROFILE_BASELINE_5_3_0  [extern]
```

The QoS default values for version 5.3.0.

In Library: **DDS_BUILTIN_QOS_LIB** (p. 685)

String-version: "Baseline.5.3.0"

7.209.2.8 DDS_PROFILE_BASELINE_6_0_0

```
const char* const DDS_PROFILE_BASELINE_6_0_0  [extern]
```

The QoS default values for version 6.0.0.

In Library: **DDS_BUILTIN_QOS_LIB** (p. 685)

String-version: "Baseline.6.0.0"

7.209.2.9 DDS_PROFILE_BASELINE_6_1_0

```
const char* const DDS_PROFILE_BASELINE_6_1_0  [extern]
```

The QoS default values for version 6.1.0.

In Library: **DDS_BUILTIN_QOS_LIB** (p. 685)

String-version: "Baseline.6.1.0"

7.209.2.10 DDS_PROFILE_BASELINE_7_0_0

```
const char* const DDS_PROFILE_BASELINE_7_0_0  [extern]
```

The QoS default values for version 7.0.0.

In Library: **DDS_BUILTIN_QOS_LIB** (p. 685)

String-version: "Baseline.7.0.0"

7.209.2.11 DDS_PROFILE_BASELINE_7_1_0

```
const char* const DDS_PROFILE_BASELINE_7_1_0  [extern]
```

The QoS default values for version 7.1.0.

In Library: **DDS_BUILTIN_QOS_LIB** (p. 685)

String-version: "Baseline.7.1.0"

7.209.2.12 DDS_PROFILE_GENERIC_COMMON

```
const char* const DDS_PROFILE_GENERIC_COMMON  [extern]
```

A common Participant base profile.

All Generic.X and Pattern.X profiles inherit from this profile.

In Library: **DDS_BUILTIN_QOS_LIB** (p. 685)

String-version: "Generic.Common"

7.209.2.13 DDS_PROFILE_GENERIC_MONITORING_COMMON

```
const char* const DDS_PROFILE_GENERIC_MONITORING_COMMON  [extern]
```

Enables RTI Monitoring Library.

Generic Base participant QoS Profile that enables RTI Monitoring Library.

Use of this QoS Profile is deprecated. It is included for backwards compatibility.

Instead of using the (deprecated) "BuiltinQosLib::Generic.Monitoring.Common", apply the QoS Snippet "BuiltinQos↩ SnippetLib::Feature.Monitoring.Enable" via composition. For example:

```
<qos_profile name="MyProfile">

    <base_name>

        <element>BuiltinQosSnippetLib::Feature.Monitoring.Enable</element>

    </base_name>

</qos_profile>
```

Legacy applications may use this QoS Profile via inheritance. To do this, create a derived QoS Profile that inherits from this one. For example:

```
<qos_profile name="MyProfile" base_name="BuiltinQosLib::Generic.StrictReliable">

    <domain_participant_qos base_name="BuiltinQosLib::Generic.Monitoring.Common">

</qos_profile>
```

In Library: **DDS_BUILTIN_QOS_LIB** (p. 685)

String-version: "Generic.Monitoring.Common"

7.209.2.14 DDS_PROFILE_GENERIC_CONNEXT_MICRO_COMPATIBILITY

```
const char* const DDS_PROFILE_GENERIC_CONNEXT_MICRO_COMPATIBILITY  [extern]
```

Sets the values necessary to communicate with RTI Connext Micro.

This profile will always represent the QoS values required for interoperability between the most recent version of RTI Connext Micro at the time of release of the most recent version of RTI Connext.

In Library: **DDS_BUILTIN_QOS_LIB** (p. 685)

String-version: "Generic.ConnextMicroCompatibility"

7.209.2.15 DDS_PROFILE_GENERIC_CONNEXT_MICRO_COMPATIBILITY_2_4_9

```
const char* const DDS_PROFILE_GENERIC_CONNEXT_MICRO_COMPATIBILITY_2_4_9  [extern]
```

Sets the values necessary to communicate with RTI Connext Micro versions 2.4.4 through at least 2.4.9.

At the time of the release of RTI Connext 5.3.0 it was not necessary to set any QoS values in order to interoperate with RTI Connext Micro. This applies to RTI Connext Micro versions 2.4.4 and later. The most recent version of RTI Connext Micro at the time of release of RTI Connext 5.3.0 was 2.4.9. There is no guarantee that this profile will interoperate with versions of RTI Connext Micro after 2.4.9.

In Library: **DDS_BUILTIN_QOS_LIB** (p. 685)

String-version: "Generic.ConnextMicroCompatibility.2.4.9"

7.209.2.16 DDS_PROFILE_GENERIC_CONNEXT_MICRO_COMPATIBILITY_2_4_3

```
const char* const DDS_PROFILE_GENERIC_CONNEXT_MICRO_COMPATIBILITY_2_4_3  [extern]
```

Sets the values necessary to communicate with RTI Connext Micro versions 2.4.3 and ealier.

RTI Connext Micro versions 2.4.3 and earlier only supported the **DDS_MANUAL_BY_TOPIC_LIVELINESS_QOS** (p. 608) LivelinessQos kind. In order to be compatible with these versions of RTI Connext Micro, the **DDSDataReader** (p. 1556) and **DDSDataWriter** (p. 1589) must have their Liveliness kind changed to this value because the default kind in RTI Connext is **DDS_AUTOMATIC_LIVELINESS_QOS** (p. 608).

In Library: **DDS_BUILTIN_QOS_LIB** (p. 685)

String-version: "Generic.ConnextMicroCompatibility.2.4.3"

7.209.2.17 DDS_PROFILE_GENERIC_OTHER_DDS_VENDOR_COMPATIBILITY

```
const char* const DDS_PROFILE_GENERIC_OTHER_DDS_VENDOR_COMPATIBILITY  [extern]
```

Sets the values necessary to interoperate with other DDS vendors.

In Library: **DDS_BUILTIN_QOS_LIB** (p. 685)

String-version: "Generic.OtherDDSVendorCompatibility"

7.209.2.18 DDS_PROFILE_GENERIC_510_TRANSPORT_COMPATIBILITY

```
const char* const DDS_PROFILE_GENERIC_510_TRANSPORT_COMPATIBILITY  [extern]
```

Sets the values necessary to interoperate with RTI Connext 5.1.0 using the UDPv6 and/or SHMEM transports.

In Library: **DDS_BUILTIN_QOS_LIB** (p. 685)

String-version: "Generic.510TransportCompatibility"

7.209.2.19 DDS_BUILTIN_QOS_LIB_EXP

```
const char* const DDS_BUILTIN_QOS_LIB_EXP  [extern]
```

A library of experimental QoS profiles.

Experimental profiles are new profiles that have been tested internally but have not gone through an extensive validation period. Therefore some of the settings may change in future releases based on customer and internal feedback. After validation, experimental profiles will be moved into the non-experimental library.

QoS Profiles in this library are deprecated. They have been moved to "BuiltinQosLib". You should use the QoS Profiles from "BuiltinQosLib" instead of the ones in "BuiltinQosLibExp". The experimental profiles are still defined here to avoid backward compatibility issues.

String-version: "BuiltinQosLibExp"

7.209.2.20 DDS_PROFILE_GENERIC_STRICT_RELIABLE

```
const char* const DDS_PROFILE_GENERIC_STRICT_RELIABLE  [extern]
```

Enables strict reliability.

Configures communication to be "strict reliable" where every sample is reliably delivered.

Combines the use of the **DDS_RELIABLE_RELIABILITY_QOS** (p. 633) for **DDS_ReliabilityQosPolicy** (p. 1295) with a **DDS_KEEP_ALL_HISTORY_QOS** (p. 604) for the **DDS_HistoryQosPolicyKind** (p. 603).

This QoS Profile also optimizes the reliability protocol setting for this configuration.

In Library: **DDS_BUILTIN_QOS_LIB** (p. 685) and **DDS_BUILTIN_QOS_LIB_EXP** (p. 689)

String-version: "Generic.StrictReliable"

7.209.2.21 DDS_PROFILE_GENERIC_KEEP_LAST_RELIABLE

```
const char* const DDS_PROFILE_GENERIC_KEEP_LAST_RELIABLE  [extern]
```

Enables keep-last reliability.

Like the Generic.StrictReliable profile, this profile ensures in-order delivery of samples. However, new data can overwrite data that has not yet been acknowledged by the reader, therefore causing possible sample loss.

In Library: **DDS_BUILTIN_QOS_LIB** (p. 685) and **DDS_BUILTIN_QOS_LIB_EXP** (p. 689)

String-version: "Generic.KeepLastReliable"

7.209.2.22 DDS_PROFILE_GENERIC_BEST_EFFORT

```
const char* const DDS_PROFILE_GENERIC_BEST_EFFORT  [extern]
```

Enables best-effort reliability kind.

This profile enables best-effort communication. No effort or resources are spent to track whether or not sent samples are received. Minimal resources are used. This is the most deterministic method of sending data since there is no indeterministic delay that can be introduced by resending data. Data samples may be lost. This setting is good for periodic data.

In Library: **DDS_BUILTIN_QOS_LIB** (p. 685) and **DDS_BUILTIN_QOS_LIB_EXP** (p. 689)

String-version: "Generic.BestEffort"

7.209.2.23 DDS_PROFILE_GENERIC_STRICT_RELIABLE_HIGH_THROUGHPUT

```
const char* const DDS_PROFILE_GENERIC_STRICT_RELIABLE_HIGH_THROUGHPUT  [extern]
```

A profile that can be used to achieve high throughput.

This QoS Profile extends the **DDS_PROFILE_GENERIC_STRICT_RELIABLE** (p. 689) QoS Profile to perform additional, fine-grained performance tuning specific to applications that send continuously streaming data. The parameters specified here add to and/or override the parameters specified in the **DDS_PROFILE_GENERIC_STRICT_RELIABLE** (p. 689) QoS Profile..

In Library: **DDS_BUILTIN_QOS_LIB** (p. 685) and **DDS_BUILTIN_QOS_LIB_EXP** (p. 689)

String-version: "Generic.StrictReliable.HighThroughput"

7.209.2.24 DDS_PROFILE_GENERIC_STRICT_RELIABLE_LOW_LATENCY

```
const char* const DDS_PROFILE_GENERIC_STRICT_RELIABLE_LOW_LATENCY  [extern]
```

A profile that can be used to achieve low latency.

This QoS Profile extends the **DDS_PROFILE_GENERIC_STRICT_RELIABLE** (p. 689) QoS Profile to perform additional, fine-grained performance tuning specific to applications that send continuously streaming data. The parameters specified here add to and/or override the parameters specified in the **DDS_PROFILE_GENERIC_STRICT_RELIABLE** (p. 689) QoS Profile.

In Library: **DDS_BUILTIN_QOS_LIB** (p. 685) and **DDS_BUILTIN_QOS_LIB_EXP** (p. 689)

String-version: "Generic.StrictReliable.LowLatency"

7.209.2.25 DDS_PROFILE_GENERIC_PARTICIPANT_LARGE_DATA

```
const char* const DDS_PROFILE_GENERIC_PARTICIPANT_LARGE_DATA  [extern]
```

A common Participant base profile to facilitate sending large data.

This is a common Participant base QoS Profile that configures 3 different flow controllers: 838, 209, and 52 Mbps that can each be used to throttle application data flow at different rates.

In Library: **DDS_BUILTIN_QOS_LIB** (p. 685) and **DDS_BUILTIN_QOS_LIB_EXP** (p. 689)

String-version: "Generic.Participant.LargeData"

7.209.2.26 DDS_PROFILE_GENERIC_PARTICIPANT_LARGE_DATA_MONITORING

```
const char* const DDS_PROFILE_GENERIC_PARTICIPANT_LARGE_DATA_MONITORING  [extern]
```

Configures Participants for large data *and* monitoring.

This is a common base Participant QoS Profile to configure Participants to both handle large data and use RTI Monitoring Library.

This QoS Profile is deprecated. It is included for backwards compatibility.

It is recommended that instead of inheriting from this QoS Profile, new applications apply the following QoS Snippets to their application-specific QoS Profiles:

Enable Monitoring

- **DDS_SNIPPET_FEATURE_MONITORING_ENABLE** (p. 710)

Enable Large Data optimizations

- **DDS_SNIPPET_OPTIMIZATION_RELIABILITY_PROTOCOL_LARGE_DATA** (p. 702)
- **DDS_SNIPPET_OPTIMIZATION_DATACACHE_LARGE_DATA_DYNAMICMEMALLOC** (p. 703) In Library↩ : **DDS_BUILTIN_QOS_LIB** (p. 685) and **DDS_BUILTIN_QOS_LIB_EXP** (p. 689)

String-version: "Generic.Participant.LargeData.Monitoring"

See also

DDS_PROFILE_GENERIC_PARTICIPANT_LARGE_DATA (p. 690)
DDS_PROFILE_GENERIC_MONITORING_COMMON (p. 687)

7.209.2.27 DDS_PROFILE_GENERIC_STRICT_RELIABLE_LARGE_DATA

```
const char* const DDS_PROFILE_GENERIC_STRICT_RELIABLE_LARGE_DATA [extern]
```

Configures endpoints for sending large data with strict reliability.

This QoS Profile extends the **DDS_PROFILE_GENERIC_PARTICIPANT_LARGE_DATA** (p. 690) QoS Profile to handle sending large samples reliably. This QoS Profile optimizes memory usage per sample within RTI Connext. You can use this QoS Profile directly, which doesn't perform any flow control, or you can use one of the three QoS Profiles below (Generic.StrictReliable.LargeData.∗Flow), which use this QoS Profile as a common base QoS Profile, but additionally enable asynchronous publication. Each of these three QoS Profiles uses one of the three flow controllers defined in **DDS_PROFILE_GENERIC_PARTICIPANT_LARGE_DATA** (p. 690) in order to throttle application data flow.

In Library: **DDS_BUILTIN_QOS_LIB** (p. 685) and **DDS_BUILTIN_QOS_LIB_EXP** (p. 689)

String-version: "Generic.StrictReliable.LargeData"

7.209.2.28 DDS_PROFILE_GENERIC_KEEP_LAST_RELIABLE_LARGE_DATA

```
const char* const DDS_PROFILE_GENERIC_KEEP_LAST_RELIABLE_LARGE_DATA [extern]
```

Configures endpoints for sending large data with keep-last reliability.

This QoS Profile is similar to the **DDS_PROFILE_GENERIC_KEEP_LAST_RELIABLE** (p. 689) QoS Profile, but also adds QoS Snippets to handle sending large samples reliably. You can use this QoS Profile directly, which doesn't perform any flow control, or you can use one of the three QoS Profiles below (Generic.KeepLastReliable.LargeData.∗Flow), which use this QoS Profile as a common base QoS Profile, but additionally enable asynchronous publication. Each of these three QoS Profiles uses one of the three flow controllers defined in **DDS_PROFILE_GENERIC_PARTICIPANT↩ _LARGE_DATA** (p. 690) in order to throttle application data flow.

In Library: **DDS_BUILTIN_QOS_LIB** (p. 685) and **DDS_BUILTIN_QOS_LIB_EXP** (p. 689)

String-version: "Generic.KeepLastReliable.LargeData"

7.209.2.29 DDS_PROFILE_GENERIC_STRICT_RELIABLE_LARGE_DATA_FAST_FLOW

```
const char* const DDS_PROFILE_GENERIC_STRICT_RELIABLE_LARGE_DATA_FAST_FLOW [extern]
```

Configures strictly reliable communication for large data with a fast flow controller.

Strictly reliable communication for large data with a 838 Mbps ($\sim$ 100 MB/sec) flow controller.

In Library: **DDS_BUILTIN_QOS_LIB** (p. 685) and **DDS_BUILTIN_QOS_LIB_EXP** (p. 689)

String-version: "Generic.StrictReliable.LargeData.FastFlow"

7.209.2.30 DDS_PROFILE_GENERIC_STRICT_RELIABLE_LARGE_DATA_MEDIUM_FLOW

```
const char* const DDS_PROFILE_GENERIC_STRICT_RELIABLE_LARGE_DATA_MEDIUM_FLOW  [extern]
```

Configures strictly reliable communication for large data with a medium flow controller.

Strictly reliable communication for large data with a 209Mbps ($\sim$ 25 MB/sec) flow controller.

In Library: **DDS_BUILTIN_QOS_LIB** (p. 685) and **DDS_BUILTIN_QOS_LIB_EXP** (p. 689)

String-version: "Generic.StrictReliable.LargeData.MediumFlow"

7.209.2.31 DDS_PROFILE_GENERIC_STRICT_RELIABLE_LARGE_DATA_SLOW_FLOW

```
const char* const DDS_PROFILE_GENERIC_STRICT_RELIABLE_LARGE_DATA_SLOW_FLOW  [extern]
```

Configures strictly reliable communication for large data with a slow flow controller.

Strictly reliable communication for large data with a 52 MB/sec ($\sim$ 6.25 MB/sec) flow controller.

In Library: **DDS_BUILTIN_QOS_LIB** (p. 685) and **DDS_BUILTIN_QOS_LIB_EXP** (p. 689)

String-version: "Generic.StrictReliable.LargeData.SlowFlow"

7.209.2.32 DDS_PROFILE_GENERIC_KEEP_LAST_RELIABLE_LARGE_DATA_FAST_FLOW

```
const char* const DDS_PROFILE_GENERIC_KEEP_LAST_RELIABLE_LARGE_DATA_FAST_FLOW  [extern]
```

Configures keep-last reliable communication for large data with a fast flow controller.

Keep-last reliable communication for large data with a 838 Mbps ($\sim$ 100 MB/sec) flow controller.

In Library: **DDS_BUILTIN_QOS_LIB** (p. 685) and **DDS_BUILTIN_QOS_LIB_EXP** (p. 689)

String-version: "Generic.KeepLastReliable.LargeData.FastFlow"

7.209.2.33 DDS_PROFILE_GENERIC_KEEP_LAST_RELIABLE_LARGE_DATA_MEDIUM_FLOW

```
const char* const DDS_PROFILE_GENERIC_KEEP_LAST_RELIABLE_LARGE_DATA_MEDIUM_FLOW  [extern]
```

Configures keep-last reliable communication for large data with a medium flow controller.

Keep-last reliable communication for large data with a 209Mbps ($\sim$ 25 MB/sec) flow controller.

In Library: **DDS_BUILTIN_QOS_LIB** (p. 685) and **DDS_BUILTIN_QOS_LIB_EXP** (p. 689)

String-version: "Generic.KeepLastReliable.LargeData.MediumFlow"

7.209.2.34 DDS_PROFILE_GENERIC_KEEP_LAST_RELIABLE_LARGE_DATA_SLOW_FLOW

```
const char* const DDS_PROFILE_GENERIC_KEEP_LAST_RELIABLE_LARGE_DATA_SLOW_FLOW  [extern]
```

Configures keep-last reliable communication for large data with a slow flow controller.

Keep-last reliable communication for large data with a 52 MB/sec ($\sim$ 6.25 MB/sec) flow controller.

In Library: **DDS_BUILTIN_QOS_LIB** (p. 685) and **DDS_BUILTIN_QOS_LIB_EXP** (p. 689)

String-version: "Generic.KeepLastReliable.LargeData.SlowFlow"

7.209.2.35 DDS_PROFILE_GENERIC_KEEP_LAST_RELIABLE_TRANSIENT_LOCAL

```
const char* const DDS_PROFILE_GENERIC_KEEP_LAST_RELIABLE_TRANSIENT_LOCAL  [extern]
```

Persists the samples of a DataWriter as long as the entity exists.

This profile extends the **DDS_PROFILE_GENERIC_KEEP_LAST_RELIABLE** (p. 689) profile, but persists the samples of a **DDSDataWriter** (p. 1589) as long as the entity exists in order to deliver them to late-joining DataReaders.

In Library: **DDS_BUILTIN_QOS_LIB** (p. 685) and **DDS_BUILTIN_QOS_LIB_EXP** (p. 689)

String-version: "Generic.KeepLastReliable.TransientLocal"

7.209.2.36 DDS_PROFILE_GENERIC_KEEP_LAST_RELIABLE_TRANSIENT

```
const char* const DDS_PROFILE_GENERIC_KEEP_LAST_RELIABLE_TRANSIENT  [extern]
```

Persists samples using RTI Persistence Service.

This profile extends the **DDS_PROFILE_GENERIC_KEEP_LAST_RELIABLE** (p. 689) profile, but persists samples using Persistence Service in order to deliver them to late-joining DataReaders.

In Library: **DDS_BUILTIN_QOS_LIB** (p. 685) and **DDS_BUILTIN_QOS_LIB_EXP** (p. 689)

String-version: "Generic.KeepLastReliable.Transient"

7.209.2.37 DDS_PROFILE_GENERIC_KEEP_LAST_RELIABLE_PERSISTENT

```
const char* const DDS_PROFILE_GENERIC_KEEP_LAST_RELIABLE_PERSISTENT  [extern]
```

Persists samples in permanent storage, like a disk, using RTI Persistence Service.

This profile extends the **DDS_PROFILE_GENERIC_KEEP_LAST_RELIABLE** (p. 689) profile, but persists samples in permanent storage, such as a disk, using Persistence Service in order to deliver them to late-joining DataReaders.

In Library: **DDS_BUILTIN_QOS_LIB** (p. 685) and **DDS_BUILTIN_QOS_LIB_EXP** (p. 689)

String-version: "Generic.KeepLastReliable.Persistent"

7.209.2.38 DDS_PROFILE_GENERIC_AUTO_TUNING

```
const char* const DDS_PROFILE_GENERIC_AUTO_TUNING  [extern]
```

Enables the Turbo Mode batching and Auto Throttle experimental features.

Turbo Mode batching adjusts the maximum number of bytes of a batch based on how frequently samples are being written. Auto Throttle auto-adjusts the speed at which a writer will write samples, based on the number of unacknowledged samples in its queue.

These features are designed to auto-adjust the publishing behavior within a system in order to achieve the best possible performance with regards to throughput and latency.

In Library: **DDS_BUILTIN_QOS_LIB** (p. 685) and **DDS_BUILTIN_QOS_LIB_EXP** (p. 689)

String-version: "Generic.AutoTuning"

7.209.2.39 DDS_PROFILE_GENERIC_MINIMAL_MEMORY_FOOTPRINT

```
const char* const DDS_PROFILE_GENERIC_MINIMAL_MEMORY_FOOTPRINT  [extern]
```

Uses a set of QoS which reduces the memory footprint of the application.

Uses a set of QoS which reduces the memory footprint of the application.

In Library: **DDS_BUILTIN_QOS_LIB** (p. 685) and **DDS_BUILTIN_QOS_LIB_EXP** (p. 689)

String-version: "Generic.MinimalMemoryFootprint"

7.209.2.40 DDS_PROFILE_GENERIC_SECURITY

```
const char* const DDS_PROFILE_GENERIC_SECURITY  [extern]
```

Loads the DDS Secure builtin plugins.

Generic Base Participant Profile that enables the builtin DDS Security Plugins.

Use of this QoS Profile is deprecated. It is included for backwards compatibility.

Instead of using the (deprecated) "BuiltinQosLib::Generic.Security", apply the QoS Snippet "BuiltinQosSnippetLib::↩ Feature.Security.Enable" via composition. For example:

```
<qos_profile name="MyProfile">

    <base_name>

        <element>BuiltinQosSnippetLib::Feature.Security.Enable</element>

    </base_name>

</qos_profile>
```

Legacy applications may use this QoS Profile via inheritance. To do this, create a derived QoS Profile that inherits from this one. For example:

```
<qos_profile name="MyProfile" base_name="BuiltinQosLib::Generic.StrictReliable">

    <domain_participant_qos base_name="BuiltinQosLib::Generic.Security">

</qos_profile>
```

In Library: **DDS_BUILTIN_QOS_LIB** (p. 685)

String-version: "Generic.Security"

7.209.2.41 DDS_PROFILE_GENERIC_MONITORING2

```
const char* const DDS_PROFILE_GENERIC_MONITORING2  [extern]
```

The default QoS profile that the DDS entities created by the RTI Monitoring Library 2.0 use.

This profile inherits from **DDS_PROFILE_GENERIC_MINIMAL_MEMORY_FOOTPRINT** (p. 695).

The following DomainParticipant QoS policy of this profile cannot be modified and it is overwritten by the RTI Monitoring Library 2.0:

- Discovery Config Built-in Channel Kind.

The following DataWriter and DataReader QoS policies of this profile cannot be modified and they are overwritten by the RTI Monitoring Library 2.0:

- Reliability Kind.
- Durability Kind.
- History Kind.
- Publish Mode Kind.
- Protocol -> RTPS Reliable Writer -> Max Heartbeat Retries.

This profile uses Topic Filters to select the DataWriter and DataReader QoS depending on the Observability Distribution Topic.

You should be using this profile or a profile inheriting from it when you configure the distribution of telemetry data using the <distribution_settings> tag under <monitoring> for the <participant_factory_qos>.

Note: This profile does not enable the use of the RTI Monitoring Library 2.0. To do that you can use the Snippet **DDS_SNIPPET_FEATURE_MONITORING2_ENABLE** (p. 711).

String-version: "Generic.Monitoring2"

See also

RTI_MONITORING_PERIODIC_TOPIC_NAME (p. 720)

RTI_MONITORING_EVENT_TOPIC_NAME (p. 720)

RTI_MONITORING_LOGGING_TOPIC_NAME (p. 721)

7.209.2.42 DDS_PROFILE_GENERIC_MONITORING2_WAN

```
const char* const DDS_PROFILE_GENERIC_MONITORING2_WAN  [extern]
```

The QoS profile that the DDS entities created by the RTI Monitoring Library 2.0 use to communicate over WAN with a Collector Service.

This profile inherits from **DDS_PROFILE_GENERIC_MONITORING2** (p. 695).

Note: This profile does not enable the use of the RTI Monitoring Library 2.0. To do that you can use the Snippet **DDS_SNIPPET_FEATURE_MONITORING2_ENABLE** (p. 711).

String-version: "Generic.Monitoring2.WAN"

See also

DDS_PROFILE_GENERIC_MONITORING2 (p. 695)

7.209.2.43 DDS_PROFILE_PATTERN_RPC

```
const char* const DDS_PROFILE_PATTERN_RPC  [extern]
```

This profile contains the default RPC and Request-Reply QoS settings.

This profile contains the QoS that is used to create the DataReaders and DataWriters in a Requester, a Replier, a Client or a Service.

Use this profile as the base profile if you need to customize the QoS settings of any of these entities.

In Library: **DDS_BUILTIN_QOS_SNIPPET_LIB** (p. 700)

String-version: "Pattern.RPC"

7.209.2.44 DDS_PROFILE_PATTERN_PERIODIC_DATA

```
const char* const DDS_PROFILE_PATTERN_PERIODIC_DATA  [extern]
```

Used for applications that expect periodic data.

This QoS Profile is intended to be used for applications that expect periodic data such as sensor data. The deadline that is set in this profile can be used to detect when DataWriters are not publishing data with the expected periodicity.

In Library: **DDS_BUILTIN_QOS_LIB** (p. 685) and **DDS_BUILTIN_QOS_LIB_EXP** (p. 689)

String-version: "Pattern.PeriodicData"

7.209.2.45 DDS_PROFILE_PATTERN_STREAMING

```
const char* const DDS_PROFILE_PATTERN_STREAMING  [extern]
```

Used for applications that stream data.

The data sent in streaming applications is commonly periodic. Therefore this profile simply inherits from the **DDS_PROFILE_PATTERN_PERIODIC_DATA** (p. 697) profile. Note: With this QoS Profile, the application may lose data, which may be acceptable in use cases such as video conferencing.

In Library: **DDS_BUILTIN_QOS_LIB** (p. 685) and **DDS_BUILTIN_QOS_LIB_EXP** (p. 689)

String-version: "Pattern.Streaming"

7.209.2.46 DDS_PROFILE_PATTERN_RELIABLE_STREAMING

```
const char* const DDS_PROFILE_PATTERN_RELIABLE_STREAMING  [extern]
```

Used for applications that stream data *and* require reliable communication.

Sometimes streaming applications require reliable communication while still tolerating some data loss. In this case, we inherit from the **DDS_PROFILE_GENERIC_KEEP_LAST_RELIABLE** (p. 689) QoS Profile and the following QoS Snippets:

- **DDS_SNIPPET_QOS_POLICY_RELIABILITY_RELIABLE** (p. 705)
- **DDS_SNIPPET_QOS_POLICY_HISTORY_KEEP_LAST_1** (p. 706)
- **DDS_SNIPPET_OPTIMIZATION_RELIABILITY_PROTOCOL_KEEP_LAST** (p. 701)

This QoS Snippet also increases the **DDS_HistoryQosPolicy::depth** (p. 1164) to reduce the probability of losing samples.

In Library: **DDS_BUILTIN_QOS_LIB** (p. 685) and **DDS_BUILTIN_QOS_LIB_EXP** (p. 689)

String-version: "Pattern.ReliableStreaming"

7.209.2.47 DDS_PROFILE_PATTERN_EVENT

```
const char* const DDS_PROFILE_PATTERN_EVENT  [extern]
```

Used for applications that handle events.

This QoS Profile can be used by applications in which samples represent events such as button pushes or alerts. When events are triggered, the system should almost always do something, meaning that you don't want the system to lose the event. This means that the system requires strictly reliable communication. To enable it, use the following QoS Snippets:

- **DDS_SNIPPET_QOS_POLICY_RELIABILITY_RELIABLE** (p. 705)
- **DDS_SNIPPET_QOS_POLICY_HISTORY_KEEP_ALL** (p. 706)
- **DDS_SNIPPET_OPTIMIZATION_RELIABILITY_PROTOCOL_KEEP_ALL** (p. 700)

Since events and alerts are critical and non-periodic data, it is important to detect situations in which communication between a **DDSDataWriter** (p. 1589) and **DDSDataReader** (p. 1556) is broken. This is why this profile sets the **DDS_LivelinessQosPolicy** (p. 1180). If the **DDSDataWriter** (p. 1589) does not assert its liveliness in a timely manner, the **DDSDataReader** (p. 1556) will report 'loss of liveliness' to the application.

In Library: **DDS_BUILTIN_QOS_LIB** (p. 685) and **DDS_BUILTIN_QOS_LIB_EXP** (p. 689)

String-version: "Pattern.Event"

7.209.2.48 DDS_PROFILE_PATTERN_ALARM_EVENT

```
const char* const DDS_PROFILE_PATTERN_ALARM_EVENT  [extern]
```

Used for applications that handle alarm events.

An alarm is a type of event; therefore this profile simply inherits from **DDS_PROFILE_PATTERN_EVENT** (p. 698).

In Library: **DDS_BUILTIN_QOS_LIB** (p. 685) and **DDS_BUILTIN_QOS_LIB_EXP** (p. 689)

String-version: "Pattern.AlarmEvent"

7.209.2.49 DDS_PROFILE_PATTERN_STATUS

```
const char* const DDS_PROFILE_PATTERN_STATUS  [extern]
```

Used for applications whose samples represent statuses.

This QoS Profile can be used by applications in which samples represent state variables whose values remain valid as long as they don't explicitly change. State variables typically do not change periodically. State variables and their values should also be available to applications that appear after the value originally changed because it is unreasonable to have to wait until the next change of state, which may be indeterminate.

Whether to use this QoS Profile or **DDS_PROFILE_PATTERN_PERIODIC_DATA** (p. 697) can often be an application choice. For example, if a DataWriter is publishing temperature sensor data, it could use the **DDS_PROFILE_PATTERN_PERIODIC_DATA** (p. 697) QoS Profile and publish the data at a fixed rate or it could use the **DDS_PROFILE_PATTERN_STATUS** (p. 699) QoS Profile and only publish the temperature when it changes more than 1 degree.

In Library: **DDS_BUILTIN_QOS_LIB** (p. 685) and **DDS_BUILTIN_QOS_LIB_EXP** (p. 689)

String-version: "Pattern.Status"

7.209.2.50 DDS_PROFILE_PATTERN_ALARM_STATUS

```
const char* const DDS_PROFILE_PATTERN_ALARM_STATUS  [extern]
```

Used for applications in which samples represent alarm statuses.

An alarm status is a type of status; therefore this QoS Profile simply inherits from **DDS_PROFILE_PATTERN_STATUS** (p. 699).

In Library: **DDS_BUILTIN_QOS_LIB** (p. 685) and **DDS_BUILTIN_QOS_LIB_EXP** (p. 689)

String-version: "Pattern.AlarmStatus"

7.209.2.51 DDS_PROFILE_PATTERN_LAST_VALUE_CACHE

```
const char* const DDS_PROFILE_PATTERN_LAST_VALUE_CACHE  [extern]
```

Used for applications that only need the last published value.

With this QoS Profile, a **DDSDataWriter** (p. 1589) will keep in its queue the last value that was published for each sample instance. Late-joining DataReaders will get that value when they join the system. This QoS Profile inherits from **DDS_PROFILE_GENERIC_KEEP_LAST_RELIABLE_TRANSIENT_LOCAL** (p. 694) because the use case requires delivery to late-joiners.

In Library: **DDS_BUILTIN_QOS_LIB** (p. 685) and **DDS_BUILTIN_QOS_LIB_EXP** (p. 689)

String-version: "Pattern.LastValueCache"

7.209.2.52 DDS_BUILTIN_QOS_SNIPPET_LIB

```
const char* const DDS_BUILTIN_QOS_SNIPPET_LIB  [extern]
```

A library of QoS Snippets.

String-version: "BuiltinQosSnippetLib"

7.209.2.53 DDS_SNIPPET_OPTIMIZATION_RELIABILITY_PROTOCOL_COMMON

```
const char* const DDS_SNIPPET_OPTIMIZATION_RELIABILITY_PROTOCOL_COMMON  [extern]
```

QoS Snippet that configures the reliability protocol with a common configuration.

The QoS Snippets starting with "Optimization.ReliabilityProtocol." provide alternative configurations of the same QoS Policies. Choose just one QoS Snippet of the alternatives when defining a new QoS Profile or QoS Snippet.

This QoS Snippet defines parameters common to a set of "alternative" QoS Snippets that configure the reliability protocol.

Modified QoS Parameters:

- **DDS_DataWriterProtocolQosPolicy::rtps_reliable_writer** (p. 896)
- **DDS_DataReaderQos** (p. 858) => **DDS_ResourceLimitsQosPolicy** (p. 1308)
- **DDS_DataReaderProtocolQosPolicy::rtps_reliable_reader** (p. 846)

This QoS Snippet configures the reliability protocol parameters for more aggressive (faster) heartbeats so that sample loss is detected and repaired faster.

This QoS Snippet also sets the **DDS_RtpsReliableWriterProtocol_t::max_heartbeat_retries** (p. 1326), which works in combination with the heartbeat rate to determine when the **DDSDataWriter** (p. 1589) considers a **DDSDataReader** (p. 1556) non-responsive. This QoS Snippet configures the **DDS_DataWriterProtocolQosPolicy::rtps_reliable_writer** (p. 896) parameters so that the **DDSDataWriter** (p. 1589) considers the **DDSDataReader** (p. 1556) to be "inactive" after 500 unresponded heartbeats.

In Library: **DDS_BUILTIN_QOS_SNIPPET_LIB** (p. 700)

String-version: "Optimization.ReliabilityProtocol.Common"

7.209.2.54 DDS_SNIPPET_OPTIMIZATION_RELIABILITY_PROTOCOL_KEEP_ALL

```
const char* const DDS_SNIPPET_OPTIMIZATION_RELIABILITY_PROTOCOL_KEEP_ALL  [extern]
```

QoS Snippet that configures the reliability protocol for KEEP_ALL.

The QoS Snippets starting with "Optimization.ReliabilityProtocol." provide alternative configurations of the same QoS Policies. Choose just one QoS Snippet of the alternatives when defining a new QoS Profile or QoS Snippet.

This QoS Snippet optimizes the **DDSDataWriter** (p. 1589) reliable protocol parameters for a reliable **DDSDataWriter** (p. 1589) with **DDS_HistoryQosPolicyKind** (p. 603) set to **DDS_KEEP_ALL_HISTORY_QOS** (p. 604).

Modified QoS Parameters:

- **DDS_DataWriterProtocolQosPolicy::rtps_reliable_writer** (p. 896)
- **DDS_DataReaderQos** (p. 858) => **DDS_ResourceLimitsQosPolicy** (p. 1308)
- **DDS_DataReaderProtocolQosPolicy::rtps_reliable_reader** (p. 846)

Note that this QoS Snippet does not configure the **DDS_ReliabilityQosPolicy** (p. 1295) or **DDS_HistoryQosPolicy** (p. 1162) QoS policies. It is intended to be used in combination with other QoS Snippets that configure those policies.

In Library: **DDS_BUILTIN_QOS_SNIPPET_LIB** (p. 700)

String-version: "Optimization.ReliabilityProtocol.KeepAll"

7.209.2.55 DDS_SNIPPET_OPTIMIZATION_RELIABILITY_PROTOCOL_KEEP_LAST

```
const char* const DDS_SNIPPET_OPTIMIZATION_RELIABILITY_PROTOCOL_KEEP_LAST  [extern]
```

QoS Snippet that configures the reliability protocol for KEEP_LAST.

The QoS Snippets starting with "Optimization.ReliabilityProtocol." provide alternative configurations of the same QoS Policies. Choose just one QoS Snippet of the alternatives when defining a new QoS Profile or QoS Snippet.

This QoS Snippet optimizes the **DDSDataWriter** (p. 1589) reliable protocol parameters for a reliable **DDSDataWriter** (p. 1589) with **DDS_HistoryQosPolicyKind** (p. 603) set to **DDS_KEEP_LAST_HISTORY_QOS** (p. 604).

Modified QoS Parameters:

- **DDS_DataWriterQos** (p. 909) => **DDS_ResourceLimitsQosPolicy** (p. 1308)
- **DDS_DataWriterProtocolQosPolicy::rtps_reliable_writer** (p. 896)
- **DDS_DataReaderQos** (p. 858) => **DDS_ResourceLimitsQosPolicy** (p. 1308)
- **DDS_DataReaderProtocolQosPolicy::rtps_reliable_reader** (p. 846)

Note that this QoS Snippet does not configure the **DDS_ReliabilityQosPolicy** (p. 1295) or **DDS_HistoryQosPolicy** (p. 1162) QoS policies. It is intended to be used in combination with other QoS Snippets that configure those policies.

In Library: **DDS_BUILTIN_QOS_SNIPPET_LIB** (p. 700)

String-version: "Optimization.ReliabilityProtocol.KeepLast"

7.209.2.56 DDS_SNIPPET_OPTIMIZATION_RELIABILITY_PROTOCOL_HIGH_RATE

```
const char* const DDS_SNIPPET_OPTIMIZATION_RELIABILITY_PROTOCOL_HIGH_RATE  [extern]
```

QoS Snippet that configures the reliability protocol for sending data at a high rate.

The QoS Snippets starting with "Optimization.ReliabilityProtocol." provide alternative configurations of the same QoS Policies. Choose just one QoS Snippet of the alternatives when defining a new QoS Profile or QoS Snippet.

This QoS Snippet optimizes the **DDSDataWriter** (p. 1589) reliable protocol parameters for a reliable **DDSDataWriter** (p. 1589) that is writing messages at high rates, especially in situations where throughput is favored over latency.

Modified QoS Parameters:

- **DDS_DataWriterProtocolQosPolicy::rtps_reliable_writer** (p. 896)
- **DDS_DataReaderQos** (p. 858) => **DDS_ResourceLimitsQosPolicy** (p. 1308)
- **DDS_DataReaderProtocolQosPolicy::rtps_reliable_reader** (p. 846)

This QoS Snippet sets a fast rate of heartbeats so that errors are detected and repaired more swiftly.

Note that to get the highest thoughout you may need to apply additional changes to the final QoS Profile. See the QoS Profile **DDS_PROFILE_GENERIC_STRICT_RELIABLE_HIGH_THROUGHPUT** (p. 690) for further information.

In Library: **DDS_BUILTIN_QOS_SNIPPET_LIB** (p. 700)

String-version: "Optimization.ReliabilityProtocol.HighRate"

7.209.2.57 DDS_SNIPPET_OPTIMIZATION_RELIABILITY_PROTOCOL_LOW_LATENCY

```
const char* const DDS_SNIPPET_OPTIMIZATION_RELIABILITY_PROTOCOL_LOW_LATENCY  [extern]
```

QoS Snippet that configures the reliability protocol for sending data at low latency.

The QoS Snippets starting with "Optimization.ReliabilityProtocol." provide alternative configurations of the same QoS Policies. Choose just one QoS Snippet of the alternatives when defining a new QoS Profile or QoS Snippet.

This QoS Snippet modifies the Reliable Protocol parameters to accomplish low latency.

Modified QoS Parameters:

- **DDS_DataWriterProtocolQosPolicy::rtps_reliable_writer** (p. 896)
- **DDS_DataReaderQos** (p. 858) => **DDS_ResourceLimitsQosPolicy** (p. 1308)
- **DDS_DataReaderProtocolQosPolicy::rtps_reliable_reader** (p. 846)

Note that to get the lowest latency you may need to apply additional changes to the final QoS Profile. See the QoS Profile **DDS_PROFILE_GENERIC_STRICT_RELIABLE_LOW_LATENCY** (p. 690) for further information.

In Library: **DDS_BUILTIN_QOS_SNIPPET_LIB** (p. 700)

String-version: "Optimization.ReliabilityProtocol.LowLatency"

7.209.2.58 DDS_SNIPPET_OPTIMIZATION_RELIABILITY_PROTOCOL_LARGE_DATA

```
const char* const DDS_SNIPPET_OPTIMIZATION_RELIABILITY_PROTOCOL_LARGE_DATA [extern]
```

QoS Snippet that configures the reliability protocol for large data.

The QoS Snippets starting with "Optimization.ReliabilityProtocol." provide alternative configurations of the same QoS Policies. Choose just one QoS Snippet of the alternatives when defining a new QoS Profile or QoS Snippet.

Modifies the Reliable Protocol parameters and Resource Limits to work better with Large Data.

Modified QoS Parameters:

- **DDS_DataWriterProtocolQosPolicy::rtps_reliable_writer** (p. 896)
- **DDS_DataReaderQos** (p. 858) => **DDS_ResourceLimitsQosPolicy** (p. 1308)
- **DDS_DataReaderProtocolQosPolicy::rtps_reliable_reader** (p. 846)

Note that, to send large data, it is recommended to apply additional changes that configure the data caches, transport buffers, etc. See, for example, **DDS_PROFILE_GENERIC_KEEP_LAST_RELIABLE_LARGE_DATA** (p. 692), and derivatives for fully functional Large Data QoS Profiles.

In Library: **DDS_BUILTIN_QOS_SNIPPET_LIB** (p. 700)

String-version: "Optimization.ReliabilityProtocol.LargeData"

7.209.2.59 DDS_SNIPPET_OPTIMIZATION_DATACACHE_LARGE_DATA_DYNAMICMEMALLOC

```
const char* const DDS_SNIPPET_OPTIMIZATION_DATACACHE_LARGE_DATA_DYNAMICMEMALLOC [extern]
```

Configures the DataWriter and DataReader caches to use dynamic memory allocation for Large Data samples.

Modified QoS Parameters:

- **DDS_DataWriterQos** (p. 909) => **DDS_ResourceLimitsQosPolicy** (p. 1308)
- **DDS_DataWriterQos** (p. 909) => **DDS_PropertyQosPolicy** (p. 1260) named "dds.data_writer.history.memory_manager.*"
- **DDS_DataReaderQos** (p. 858) => **DDS_ResourceLimitsQosPolicy** (p. 1308)
- **DDS_DataReaderResourceLimitsQosPolicy** (p. 867)
- **DDS_DataReaderQos** (p. 858) => **DDS_PropertyQosPolicy** (p. 1260) named "dds.data_reader.history.memory_manager.*"

This configuration is needed to handle data that contains unbounded sequences or strings. This QoS Snippet is also recommended if samples can have very different sizes and the bigger samples can be very large.

If dynamic memory allocation is not used for the larger samples, then all samples are allocated to their maximum size which can consume a lot of resources.

In Library: **DDS_BUILTIN_QOS_SNIPPET_LIB** (p. 700)

String-version: "Optimization.DataCache.LargeData.DynamicMemAlloc"

7.209.2.60 DDS_SNIPPET_OPTIMIZATION_DISCOVERY_COMMON

```
const char* const DDS_SNIPPET_OPTIMIZATION_DISCOVERY_COMMON  [extern]
```

QoS Snippet that optimizes discovery with a common configuration.

Optimizes the **DDS_DomainParticipantQos** (p. 971) to detect faster discovery changes. This QoS Snippet increases the speed moderately so that it fits the normal scenarios.

Modified QoS Parameters:

- **DDS_DiscoveryConfigQosPolicy::participant_liveliness_lease_duration** (p. 937)
- **DDS_DiscoveryConfigQosPolicy::participant_liveliness_assert_period** (p. 937)
- **DDS_DiscoveryConfigQosPolicy::max_liveliness_loss_detection_period** (p. 938)
- **DDS_DiscoveryConfigQosPolicy::initial_participant_announcements** (p. 939)
- **DDS_DiscoveryConfigQosPolicy::publication_writer** (p. 943)
- **DDS_DiscoveryConfigQosPolicy::subscription_writer** (p. 945)

In Library: **DDS_BUILTIN_QOS_SNIPPET_LIB** (p. 700)

String-version: "Optimization.Discovery.Common"

7.209.2.61 DDS_SNIPPET_OPTIMIZATION_DISCOVERY_PARTICIPANT_COMPACT

```
const char* const DDS_SNIPPET_OPTIMIZATION_DISCOVERY_PARTICIPANT_COMPACT  [extern]
```

QoS Snippet that optimizes the Participant QoS to send less discovery information.

Modified QoS Parameters:

- **DDS_DomainParticipantQos** (p. 971) => **DDS_PropertyQosPolicy** (p. 1260) named "dds.participant.inter_↩ participant.*"
- **DDS_DomainParticipantQos** (p. 971) => **DDS_PropertyQosPolicy** (p. 1260) named "dds.sys_info.*"

In Library: **DDS_BUILTIN_QOS_SNIPPET_LIB** (p. 700)

String-version: "Optimization.Discovery.Participant.Compact"

7.209.2.62 DDS_SNIPPET_OPTIMIZATION_DISCOVERY_ENDPOINT_FAST

```
const char* const DDS_SNIPPET_OPTIMIZATION_DISCOVERY_ENDPOINT_FAST  [extern]
```

QoS Snippet that optimizes the Endpoint Discovery to be faster.

This is useful when using security, to prevent a noticeable delay.

Modified QoS Parameters:

- **DDS_DiscoveryConfigQosPolicy::publication_writer** (p. 943)
- **DDS_DiscoveryConfigQosPolicy::subscription_writer** (p. 945)

In Library: **DDS_BUILTIN_QOS_SNIPPET_LIB** (p. 700)

String-version: "Optimization.Discovery.Endpoint.Fast"

7.209.2.63 DDS_SNIPPET_OPTIMIZATION_TRANSPORT_LARGE_BUFFERS

```
const char* const DDS_SNIPPET_OPTIMIZATION_TRANSPORT_LARGE_BUFFERS  [extern]
```

QoS Snippet that increases the Participant default buffer that shmem and udpv4 use.

This is useful when using Large Data

Modified QoS Parameters:

- **DDS_ReceiverPoolQosPolicy** (p. 1293)
- **DDS_TransportBuiltinQosPolicy** (p. 1408)

In Library: **DDS_BUILTIN_QOS_SNIPPET_LIB** (p. 700)

String-version: "Optimization.Transport.LargeBuffers"

7.209.2.64 DDS_SNIPPET_QOS_POLICY_RELIABILITY_RELIABLE

```
const char* const DDS_SNIPPET_QOS_POLICY_RELIABILITY_RELIABLE  [extern]
```

QoS Snippet that sets RELIABILITY QoS to RELIABLE.

This also configures a blocking time in case the **DDSDataWriter** (p. 1589) writes faster than the DataReaders can accommodate.

Modified QoS Parameters:

- **DDS_DataWriterQos** (p. 909) => **DDS_ReliabilityQosPolicy** (p. 1295)
- **DDS_DataReaderQos** (p. 858) => **DDS_ReliabilityQosPolicy** (p. 1295)

Note that by itself enabling reliability does not ensure that every sample written is delivered to the DataReaders. This is because the **DDSDataWriter** (p. 1589) and/or **DDSDataReader** (p. 1556) can be configured to override samples in its cache based on the configuration of the **DDS_HistoryQosPolicy** (p. 1162) QoS policy.

To ensure delivery of every sample (at the expense of potentially blocking the **DDSDataWriter** (p. 1589)), use the QoS Profile **DDS_PROFILE_GENERIC_STRICT_RELIABLE** (p. 689) or one of the derived QoS Profiles.

In Library: **DDS_BUILTIN_QOS_SNIPPET_LIB** (p. 700)

String-version: "QosPolicy.Reliability.Reliable"

7.209.2.65 DDS_SNIPPET_QOS_POLICY_RELIABILITY_BEST_EFFORT

```
const char* const DDS_SNIPPET_QOS_POLICY_RELIABILITY_BEST_EFFORT  [extern]
```

QoS Snippet that sets RELIABILITY QoS to BEST_EFFORT.

Modified QoS Parameters:

- **DDS_DataWriterQos** (p. 909) => **DDS_ReliabilityQosPolicy** (p. 1295)
- **DDS_DataReaderQos** (p. 858) => **DDS_ReliabilityQosPolicy** (p. 1295)

With best-effort, there are no resources spent to confirm delivery of samples nor repairs of any samples that may be lost.

Best-effort communication reduces jitter; therefore, the delay between sending data and receiving it is more deterministic for the samples that are actually received. Best-effort is good for periodic data where it may be better to get the next value than to wait for the previous one to be repaired.

In Library: **DDS_BUILTIN_QOS_SNIPPET_LIB** (p. 700)

String-version: "QosPolicy.Reliability.BestEffort"

7.209.2.66 DDS_SNIPPET_QOS_POLICY_HISTORY_KEEP_LAST_1

```
const char* const DDS_SNIPPET_QOS_POLICY_HISTORY_KEEP_LAST_1  [extern]
```

QoS Snippet that sets HISTORY QoS to KEEP_LAST kind with depth 1.

Modified QoS Parameters:

- **DDS_DataWriterQos** (p. 909) => **DDS_HistoryQosPolicy** (p. 1162)
- **DDS_DataReaderQos** (p. 858) => **DDS_HistoryQosPolicy** (p. 1162)

In Library: **DDS_BUILTIN_QOS_SNIPPET_LIB** (p. 700)

String-version: "QosPolicy.History.KeepLast_1"

7.209.2.67 DDS_SNIPPET_QOS_POLICY_HISTORY_KEEP_ALL

```
const char* const DDS_SNIPPET_QOS_POLICY_HISTORY_KEEP_ALL  [extern]
```

QoS Snippet that sets HISTORY QosPolicy to KEEP_ALL kind.

Modified QoS Parameters:

- **DDS_DataWriterQos** (p. 909) => **DDS_HistoryQosPolicy** (p. 1162)
- **DDS_DataReaderQos** (p. 858) => **DDS_HistoryQosPolicy** (p. 1162)

In Library: **DDS_BUILTIN_QOS_SNIPPET_LIB** (p. 700)

String-version: "QosPolicy.History.KeepAll"

7.209.2.68 DDS_SNIPPET_QOS_POLICY_PUBLISH_MODE_ASYNCHRONOUS

```
const char* const DDS_SNIPPET_QOS_POLICY_PUBLISH_MODE_ASYNCHRONOUS  [extern]
```

QoS Snippet that sets PUBLISH_MODE QosPolicy to ASYNCHRONOUS kind.

Modified QoS Parameters:

- **DDS_PublishModeQosPolicy** (p. 1279)

Asynchronous Publish mode decouples the application thread that calls the **DDSDataWriter** (p. 1589) "write" operation from the thread used to send the data on the network. See https://community.rti.↩ com/glossary/asynchronous-writer

In Library: **DDS_BUILTIN_QOS_SNIPPET_LIB** (p. 700)

String-version: "QosPolicy.PublishMode.Asynchronous"

7.209.2.69 DDS_SNIPPET_QOS_POLICY_DURABILITY_TRANSIENT_LOCAL

```
const char* const DDS_SNIPPET_QOS_POLICY_DURABILITY_TRANSIENT_LOCAL  [extern]
```

QoS Snippet that sets DURABILITY QosPolicy to TRANSIENT_LOCAL kind.

Modified QoS Parameters:

- **DDS_DataWriterQos** (p. 909) => **DDS_DurabilityQosPolicy** (p. 1005)
- **DDS_DataReaderQos** (p. 858) => **DDS_DurabilityQosPolicy** (p. 1005)

DataWriters will store and send previously published DDS samples for delivery to newly discovered DataReaders as long as the **DDSDataWriter** (p. 1589) still exists. For this setting to be effective, you must also set the **DDS_ReliabilityQos**↩ **PolicyKind** (p. 632) to **DDS_RELIABLE_RELIABILITY_QOS** (p. 633) (not Best Effort). Which particular DDS samples are kept depends on other QoS settings such as **DDS_HistoryQosPolicy** (p. 1162) and **DDS_ResourceLimitsQos**↩ **Policy** (p. 1308).

In Library: **DDS_BUILTIN_QOS_SNIPPET_LIB** (p. 700)

String-version: "QosPolicy.Durability.TransientLocal"

7.209.2.70 DDS_SNIPPET_QOS_POLICY_DURABILITY_TRANSIENT

```
const char* const DDS_SNIPPET_QOS_POLICY_DURABILITY_TRANSIENT  [extern]
```

QoS Snippet that sets DURABILITY QosPolicy to TRANSIENT kind.

Modified QoS Parameters:

- **DDS_DataWriterQos** (p. 909) => **DDS_DurabilityQosPolicy** (p. 1005)
- **DDS_DataReaderQos** (p. 858) => **DDS_DurabilityQosPolicy** (p. 1005)

RTI Connext will store previously published DDS samples in memory using Persistence Service, which will send the stored data to newly discovered DataReaders. Which particular DDS samples are kept and sent by Persistence Service depends on the **DDS_HistoryQosPolicy** (p. 1162) and **DDS_ResourceLimitsQosPolicy** (p. 1308) of the Persistence Service DataWriters. These QosPolicies can be configured in the Persistence Service configuration file or through the **DDS_DurabilityQosPolicyKind** (p. 598) of the DataWriters configured with **DDS_TRANSIENT_DURABILITY_QOS** (p. 599).

You need a Persistence Service instance running to use this behavior.

In Library: **DDS_BUILTIN_QOS_SNIPPET_LIB** (p. 700)

String-version: "QosPolicy.Durability.Transient"

7.209.2.71 DDS_SNIPPET_QOS_POLICY_DURABILITY_PERSISTENT

```
const char* const DDS_SNIPPET_QOS_POLICY_DURABILITY_PERSISTENT  [extern]
```

QoS Snippet that sets DURABILITY QosPolicy to PERSISTENT kind.

Modified QoS Parameters:

- **DDS_DataWriterQos** (p. 909) => **DDS_DurabilityQosPolicy** (p. 1005)
- **DDS_DataReaderQos** (p. 858) => **DDS_DurabilityQosPolicy** (p. 1005)

RTI Connext will store previously published DDS samples in permanent storage, like a disk, using Persistence Service, which will send the stored data to newly discovered DataReaders. Which particular DDS samples are kept and sent by Persistence Service depends on the **DDS_HistoryQosPolicy** (p. 1162) and **DDS_ResourceLimitsQosPolicy** (p. 1308) in the Persistence Service DataWriters. These QosPolicies can be configured in the Persistence Service configuration file or through the **DDS_DurabilityQosPolicyKind** (p. 598) of the DataWriters configured with **DDS_PERSISTENT_↩ DURABILITY_QOS** (p. 599).

You need a Persistence Service instance running to use this behavior.

In Library: **DDS_BUILTIN_QOS_SNIPPET_LIB** (p. 700)

String-version: "QosPolicy.Durability.Persistent"

7.209.2.72 DDS_SNIPPET_QOS_POLICY_BATCHING_ENABLE

```
const char* const DDS_SNIPPET_QOS_POLICY_BATCHING_ENABLE  [extern]
```

QoS Snippet that sets BATCH QosPolicy to true.

Modified QoS Parameters:

- **DDS_BatchQosPolicy** (p. 810)

This QoS Snippet specifies and configures the mechanism that allows RTI Connext to collect multiple user data DDS samples to be sent in a single network packet, to take advantage of the efficiency of sending larger packets and thus increase effective throughput.

In Library: **DDS_BUILTIN_QOS_SNIPPET_LIB** (p. 700)

String-version: "QosPolicy.Batching.Enable"

7.209.2.73 DDS_SNIPPET_FEATURE_FLOW_CONTROLLER_838MBPS

```
const char* const DDS_SNIPPET_FEATURE_FLOW_CONTROLLER_838MBPS  [extern]
```

QoS Snippet that configures and set a FlowController of 838 Mbps.

Defines a **DDSFlowController** (p. 1736) and configures the **DDS_DataWriterQos** (p. 909) with it.

This is a **DDSFlowController** (p. 1736) of 838 Mbps ($\sim$ 100 MB/sec)

Modified QoS Parameters:

- **DDS_DomainParticipantQos** (p. 971) => **DDS_PropertyQosPolicy** (p. 1260) named "dds.flow_controller.↩ token_bucket.*"
- **DDS_PublishModeQosPolicy** (p. 1279)

In Library: **DDS_BUILTIN_QOS_SNIPPET_LIB** (p. 700)

String-version: "Feature.FlowController.838Mbps"

7.209.2.74 DDS_SNIPPET_FEATURE_FLOW_CONTROLLER_209MBPS

```
const char* const DDS_SNIPPET_FEATURE_FLOW_CONTROLLER_209MBPS  [extern]
```

QoS Snippet that configures and sets a FlowController of 209 Mbps.

Defines a **DDSFlowController** (p. 1736) and configures the **DDS_DataWriterQos** (p. 909) with it.

This is a **DDSFlowController** (p. 1736) of 209 Mbps ($\sim$ 25 MB/sec)

Modified QoS Parameters:

- **DDS_DomainParticipantQos** (p. 971) => **DDS_PropertyQosPolicy** (p. 1260) named "dds.flow_controller.↩ token_bucket.*"
- **DDS_PublishModeQosPolicy** (p. 1279)

In Library: **DDS_BUILTIN_QOS_SNIPPET_LIB** (p. 700)

String-version: "Feature.FlowController.209Mbps"

7.209.2.75 DDS_SNIPPET_FEATURE_FLOW_CONTROLLER_52MBPS

```
const char* const DDS_SNIPPET_FEATURE_FLOW_CONTROLLER_52MBPS  [extern]
```

QoS Snippet that configures and sets a FlowController of 52 Mbps.

Defines a **DDSFlowController** (p. 1736) and configures the **DDS_DataWriterQos** (p. 909) with it.

This is a **DDSFlowController** (p. 1736) of 52 Mbps ($\sim$ 6.25 MB/sec)

Modified QoS Parameters:

- **DDS_DomainParticipantQos** (p. 971) => **DDS_PropertyQosPolicy** (p. 1260) named "dds.flow_controller.↩ token_bucket.∗"
- **DDS_PublishModeQosPolicy** (p. 1279)

In Library: **DDS_BUILTIN_QOS_SNIPPET_LIB** (p. 700)

String-version: "Feature.FlowController.52Mbps"

7.209.2.76 DDS_SNIPPET_FEATURE_AUTO_TUNING_ENABLE

```
const char* const DDS_SNIPPET_FEATURE_AUTO_TUNING_ENABLE  [extern]
```

QoS Snippet that enables auto_throttle and turbo_mode to true.

Sets the **DDS_DomainParticipantQos** (p. 971) properties to enable auto_throttle and turbo_mode to true.

Modified QoS Parameters:

- **DDS_DomainParticipantQos** (p. 971) => **DDS_PropertyQosPolicy** (p. 1260) named "dds.domain_↩ participant.auto_throttle"
- **DDS_DataWriterQos** (p. 909) => **DDS_PropertyQosPolicy** (p. 1260) named "dds.data_writer.auto_throttle.↩ enable"
- **DDS_DataWriterQos** (p. 909) => **DDS_PropertyQosPolicy** (p. 1260) named "dds.data_writer.enable_turbo_↩ mode"

The domain_participant.auto_throttle configures the **DDSDomainParticipant** (p. 1619) to gather internal measurements (during **DDSDomainParticipant** (p. 1619) creation) that are required for the Auto Throttle feature. This allows Data↩ Writers belonging to this **DDSDomainParticipant** (p. 1619) to use the Auto Throttle feature.

The turbo_mode adjusts the batch max_data_bytes based on how frequently the **DDSDataWriter** (p. 1589) writes data.

Data_writer.auto_throttle enables automatic throttling in the **DDSDataWriter** (p. 1589) so it can automatically adjust the writing rate and the send window size; this minimizes the need for repairing DDS samples and improves latency.

In Library: **DDS_BUILTIN_QOS_SNIPPET_LIB** (p. 700)

String-version: "Feature.AutoTuning.Enable"

7.209.2.77 DDS_SNIPPET_FEATURE_MONITORING_ENABLE

```
const char* const DDS_SNIPPET_FEATURE_MONITORING_ENABLE  [extern]
```

QoS Snippet that enables the use of the RTI Monitoring Library.

To enable the use of RTI Monitoring Library apply this QoS Snippet to the QoS Profile used to create your **DDSDomain↩ Participant** (p. 1619). For example:

```
 <qos_profile name="MyProfile">

    <base_name>

        <element>BuiltinQosLib::Feature.Monitoring.Enable</element>

    </base_name>

</qos_profile>
```

In Library: **DDS_BUILTIN_QOS_SNIPPET_LIB** (p. 700)

String-version: "Feature.Monitoring.Enable"

7.209.2.78 DDS_SNIPPET_FEATURE_MONITORING2_ENABLE

```
const char* const DDS_SNIPPET_FEATURE_MONITORING2_ENABLE  [extern]
```

QoS Snippet that enables the use of the RTI Monitoring Library 2.0.

QoS Snippet to enable the use of the RTI Monitoring Library 2.0 with a dedicated DomainParticipant publishing telemetry data from your application in domain ID 2. All metrics are enabled for all resources in this profile.

To enable the use of RTI Monitoring Library 2.0, apply this QoS Snippet to the QoS Profile used to create your Domain↩ ParticipantFactory. For example:

```
<qos_profile name="MyProfile" is_default_participant_factory_profile="true">
  <base_name>
    <element>BuiltinQosSnippetLib::Feature.Monitoring2.Enable</element>
  </base_name>
</qos_profile>
```

In Library: **DDS_BUILTIN_QOS_SNIPPET_LIB** (p. 700)

String-version: "Feature.Monitoring2.Enable"

7.209.2.79 DDS_SNIPPET_FEATURE_SECURITY_ENABLE

```
const char* const DDS_SNIPPET_FEATURE_SECURITY_ENABLE  [extern]
```

QoS Snippet that enables security using the Builtin Security Plugins.

To enable the use of the Builtin DDS Security Library apply this QoS Snippet to the QoS Profile used to create your **DDSDomainParticipant** (p. 1619). For example:

```
 <qos_profile name="MyProfile">

    <base_name>

        <element>BuiltinQosLib::Feature.Security.Enable</element>

    </base_name>

</qos_profile>
```

In Library: **DDS_BUILTIN_QOS_SNIPPET_LIB** (p. 700)

String-version: "Feature.Security.Enable"

7.209.2.80 DDS_SNIPPET_FEATURE_TOPIC_QUERY_ENABLE

```
const char* const DDS_SNIPPET_FEATURE_TOPIC_QUERY_ENABLE  [extern]
```

QoS Snippet that enables Topic Query.

To enable the use of the RTI Connext **DDSTopicQuery** (p. 1895) feature, apply this QoS Snippet to the QoS Profile used to create your **DDSDataWriter** (p. 1589). For example:

```
<qos_profile name="MyProfile">

    <base_name>

        <element>BuiltinQosLib::Feature.TopicQuery.Enable</element>

    </base_name>

</qos_profile>
```

For more information on Topic Query see the "Topic Queries" chapter in the `User's Manual`.

In Library: **DDS_BUILTIN_QOS_SNIPPET_LIB** (p. 700)

String-version: "Feature.TopicQuery.Enable"

7.209.2.81 DDS_SNIPPET_FEATURE_TYPE_SUPPORT_V1_ENABLE

```
const char* const DDS_SNIPPET_FEATURE_TYPE_SUPPORT_V1_ENABLE  [extern]
```

QoS Snippet that enables TypeObject V1 propagation.

To enable TypeObject V1 propagation apply this QoS Snippet to the QoS Profile used to create your **DDSDomain↩ Participant** (p. 1619). For example:

```
<qos_profile name="MyProfile">

    <base_name>

        <element>BuiltinQosLib::Feature.TypeSupport.V1.Enable</element>

    </base_name>

</qos_profile>
```

7.209.2.82 DDS_SNIPPET_FEATURE_TYPE_SUPPORT_V1_DISABLE

```
const char* const DDS_SNIPPET_FEATURE_TYPE_SUPPORT_V1_DISABLE  [extern]
```

QoS Snippet that disables TypeObject V1 propagation.

To disable TypeObject V1 propagation apply this QoS Snippet to the QoS Profile used to create your **DDSDomain↩ Participant** (p. 1619). For example:

```
<qos_profile name="MyProfile">

    <base_name>

        <element>BuiltinQosLib::Feature.TypeSupport.V1.Disable</element>

    </base_name>

</qos_profile>
```

7.209.2.83 DDS_SNIPPET_FEATURE_TYPE_SUPPORT_V2_ENABLE

```
const char* const DDS_SNIPPET_FEATURE_TYPE_SUPPORT_V2_ENABLE  [extern]
```

QoS Snippet that enables TypeObject V2 propagation.

To enable TypeObject V2 propagation apply this QoS Snippet to the QoS Profile used to create your **DDSDomain↩ Participant** (p. 1619). For example:

```
 <qos_profile name="MyProfile">

    <base_name>

        <element>BuiltinQosLib::Feature.TypeSupport.V2.Enable</element>

    </base_name>

</qos_profile>
```

7.209.2.84 DDS_SNIPPET_FEATURE_TYPE_SUPPORT_V2_DISABLE

```
const char* const DDS_SNIPPET_FEATURE_TYPE_SUPPORT_V2_DISABLE  [extern]
```

QoS Snippet that disables TypeObject V2 propagation.

To disable TypeObject V2 propagation apply this QoS Snippet to the QoS Profile used to create your **DDSDomain↩ Participant** (p. 1619). For example:

```
 <qos_profile name="MyProfile">

    <base_name>

        <element>BuiltinQosLib::Feature.TypeSupport.V2.Disable</element>

    </base_name>

</qos_profile>
```

7.209.2.85 DDS_SNIPPET_TRANSPORT_TCP_LAN_CLIENT

```
const char* const DDS_SNIPPET_TRANSPORT_TCP_LAN_CLIENT  [extern]
```

QoS Snippet that configures a TCP LAN Client over DDS.

This QoS Snippet sets all the mandatory properties; however the final QoS Profile requires additional configuration.

The values of the initial_peers and the property dds.transport.TCPv4.tcp1.server_bind_port are incorrect (just sample strings). Therefore, they must be modified:

- initial_peers: should point to the remote client IP and port.
- server_bind_port: is the port this application will be using.

This modification should be done in the QoS Profile that will be used to create the **DDSEntity** (p. 1731). These new values will overwrite the current invalid values.

In Library: **DDS_BUILTIN_QOS_SNIPPET_LIB** (p. 700)

String-version: "Transport.TCP.LAN.Client"

7.209.2.86 DDS_SNIPPET_TRANSPORT_TCP_WAN_SYMMETRIC_CLIENT

```
const char* const DDS_SNIPPET_TRANSPORT_TCP_WAN_SYMMETRIC_CLIENT  [extern]
```

QoS Snippet that configures a symmetric WAN TCP Client over DDS.

This QoS Snippet sets all the mandatory properties; however the final QoS Profile requires additional configuration.

The values of the properties dds.transport.TCPv4.tcp1.public_address and dds.transport.TCPv4.tcp1.server_bind_port are incorrect (just sample strings). Also the initial_peers information is not correct. Therefore, the following must be modified:

- initial_peers: should point to the remote client IP and port.
- public_address: public IP address where this application can be reached.
- server_bind_port: port this application will be using.

This modification should be done in the QoS Profile that will be used to create the **DDSEntity** (p. 1731). These new values will overwrite the current invalid values.

In Library: **DDS_BUILTIN_QOS_SNIPPET_LIB** (p. 700)

String-version: "Transport.TCP.WAN.Symmetric.Client"

7.209.2.87 DDS_SNIPPET_TRANSPORT_TCP_WAN_ASYMMETRIC_SERVER

```
const char* const DDS_SNIPPET_TRANSPORT_TCP_WAN_ASYMMETRIC_SERVER  [extern]
```

QoS Snippet that an asymmetric WAN TCP Server over DDS.

This QoS Snippet sets all the mandatory properties; however the final QoS Profile requires additional configuration.

The values of the properties dds.transport.TCPv4.tcp1.public_address and dds.transport.TCPv4.tcp1.server_bind_port are incorrect (just sample strings). Therefore, they must be modified to the corresponding public_address and port_↩ number. This modification should be done in the QoS Profile that will be used to create the **DDSEntity** (p. 1731). These new values will overwrite the current invalid values.

In Library: **DDS_BUILTIN_QOS_SNIPPET_LIB** (p. 700)

String-version: "Transport.TCP.WAN.Asymmetric.Server"

7.209.2.88 DDS_SNIPPET_TRANSPORT_TCP_WAN_ASYMMETRIC_CLIENT

```
const char* const DDS_SNIPPET_TRANSPORT_TCP_WAN_ASYMMETRIC_CLIENT  [extern]
```

QoS Snippet that configures an asymmetric WAN TCP Client over DDS.

This QoS Snippet sets all the mandatory properties; however the final QoS Profile requires additional configuration.

The value of discovery.initial_peers and public_ip have to match the values set on the Server side (**DDS_SNIPPET_**↩ **TRANSPORT_TCP_WAN_ASYMMETRIC_SERVER** (p. 714)). This modification should be done in the QoS Profile that will be used to create the **DDSEntity** (p. 1731). This new value will overwrite the current invalid value.

In Library: **DDS_BUILTIN_QOS_SNIPPET_LIB** (p. 700)

String-version: "Transport.TCP.WAN.Asymmetric.Client"

7.209.2.89 DDS_SNIPPET_TRANSPORT_UDP_AVOID_IP_FRAGMENTATION

```
const char* const DDS_SNIPPET_TRANSPORT_UDP_AVOID_IP_FRAGMENTATION [extern]
```

QoS Snippet that configures RTI Connext and the UDP built-in transports (UDPv4, UDPv6, UDPv4_WAN) to avoid IP fragmentation.

For WAN communications and, in general, for communications in third party networks, it is not a good idea to rely on IP fragmentation. IP fragmentation causes significant issues in UDP, where there is no integrated support for a path MTU (maximum transmission unit) discovery protocol as there is in TCP.

This snippet provides a way to avoid IP fragmentation in Connext applications using the built-in UDP transports. Instead, Connext will be responsible for fragmentation, which is done at the RTPS level.

Among other changes, this configuration changes the transport MTU (message_size_max) to be 1400 bytes. Notice that this change will affect other transports such as SHMEM since Connext chooses the minimum transport MTU across all enabled transports to determine the maximum size of outgoing RTPS messages.

In Library: **DDS_BUILTIN_QOS_SNIPPET_LIB** (p. 700)

String-version: "Transport.UDP.AvoidIPFragmentation"

7.209.2.90 DDS_SNIPPET_TRANSPORT_UDP_WAN

```
const char* const DDS_SNIPPET_TRANSPORT_UDP_WAN [extern]
```

QoS Snippet that enables the RTI Real-Time WAN Transport (UDPv4_WAN).

The snippet disables all the other built-in transports and avoids the use of IP fragmentation.

For WAN communications and, in general, for communications in third-party networks, it is not a good idea to rely on IP fragmentation. IP fragmentation causes significant issues in UDP, where there is no integrated support for a path MTU (maximum transmission unit) discovery protocol as there is in TCP.

In Library: **DDS_BUILTIN_QOS_SNIPPET_LIB** (p. 700)

String-version: "Transport.UDP.WAN"

7.209.2.91 DDS_SNIPPET_COMPATIBILITY_CONNEXT_MICRO_VERSION_2_4_3

```
const char* const DDS_SNIPPET_COMPATIBILITY_CONNEXT_MICRO_VERSION_2_4_3 [extern]
```

QoS Snippet that configures sets LIVELINESS QoS to be compatible with RTI Connext Micro 2.4.3.

QoS Snippet that sets the **DDS_DataReaderQos** (p. 858) and **DDS_DataWriterQos** (p. 909) **DDS_LivelinessQos↩ PolicyKind** (p. 607) to **DDS_MANUAL_BY_TOPIC_LIVELINESS_QOS** (p. 608) It also disables the built-in shared memory transport.

Modified QoS Parameters:

- **DDS_DataReaderQos** (p. 858) => **DDS_LivelinessQosPolicy** (p. 1180)
- **DDS_DataWriterQos** (p. 909) => **DDS_LivelinessQosPolicy** (p. 1180)
- **DDS_TransportBuiltinQosPolicy** (p. 1408)

RTI Connext Micro versions 2.4.3 and earlier only supported **DDS_MANUAL_BY_TOPIC_LIVELINESS_QOS** (p. 608) **DDS_LivelinessQosPolicyKind** (p. 607). In order to be compatible with RTI Connext Micro 2.4.3, the **DDSDataReader** (p. 1556) and **DDSDataWriter** (p. 1589) must have their **DDS_LivelinessQosPolicyKind** (p. 607) changed to this value because the default kind in RTI Connext is **DDS_AUTOMATIC_LIVELINESS_QOS** (p. 608).

In Library: **DDS_BUILTIN_QOS_SNIPPET_LIB** (p. 700)

String-version: "Compatibility.ConnextMicro.Version243"

7.209.2.92 DDS_SNIPPET_COMPATIBILITY_OTHER_DDS_VENDORS_ENABLE

```
const char* const DDS_SNIPPET_COMPATIBILITY_OTHER_DDS_VENDORS_ENABLE  [extern]
```

QoS Snippet that configures RTI Connext to interoperate with other DDS vendors.

In Library: **DDS_BUILTIN_QOS_SNIPPET_LIB** (p. 700)

String-version: "Compatibility.OtherDDSVendor.Enable"

7.209.2.93 DDS_SNIPPET_5_1_0_TRANSPORT_ENABLE

```
const char* const DDS_SNIPPET_5_1_0_TRANSPORT_ENABLE  [extern]
```

QoS Snippet that configures RTI Connext to interoperate with RTI Connext 5.1.0 and below for UDPv6 and SHMEM transports.

In Library: **DDS_BUILTIN_QOS_SNIPPET_LIB** (p. 700)

String-version: "Compatibility.510Transport.Enable"

7.210 DomainParticipantConfigParams

<<***extension***>> *(p. 432)* **DDS_DomainParticipantConfigParams_t** (p. 964)

Classes

- struct **DDS_DomainParticipantConfigParams_t**

  <<***extension***>> *(p. 432) Input paramaters for creating a participant from configuration. It allows to modify or override some of the properties of the entities defined in the configuration.*

Macros

- #define **DDS_DomainParticipantConfigParams_t_INITIALIZER**

  <<***extension***>> *(p. 432) Initializer for new* ***DDS_DomainParticipantConfigParams_t*** *(p. 964).*

Variables

- const int **DDS_DOMAIN_ID_USE_XML_CONFIG**

  <<***extension***>> *(p. 432) Special value to be used with* ***DDS_DomainParticipantConfigParams_t*** *(p. 964) to indicate that a participant is created using the domain ID specified in the participant configuration.*
- const char * **DDS_ENTITY_NAME_USE_XML_CONFIG**

  <<***extension***>> *(p. 432) Special value to be used with* ***DDS_DomainParticipantConfigParams_t*** *(p. 964) to indicate that a participant created is with an autogenerated entity name.*
- const char * **DDS_QOS_ELEMENT_NAME_USE_XML_CONFIG**

  <<***extension***>> *(p. 432) Special value to be used with* ***DDS_DomainParticipantConfigParams_t*** *(p. 964) to indicate that entities are created from the QoS profile specified in the participant configuration.*

7.210.1 Detailed Description

<<***extension***>> *(*p. *432)* **DDS_DomainParticipantConfigParams_t** (p. 964)

7.210.2 Macro Definition Documentation

7.210.2.1 DDS_DomainParticipantConfigParams_t_INITIALIZER

```
#define DDS_DomainParticipantConfigParams_t_INITIALIZER
```

<<***extension***>> *(*p. *432)* Initializer for new **DDS_DomainParticipantConfigParams_t** (p. 964).

No memory is allocated. New **DDS_DomainParticipantConfigParams_t** (p. 964) instances stored in the stack should be initialized with this value before they are passed to any methods. .

7.210.3 Variable Documentation

7.210.3.1 DDS_DOMAIN_ID_USE_XML_CONFIG

```
const int DDS_DOMAIN_ID_USE_XML_CONFIG  [extern]
```

<<***extension***>> *(*p. *432)* Special value to be used with **DDS_DomainParticipantConfigParams_t** (p. 964) to indicate that a participant is created using the domain ID specified in the participant configuration.

This variable contains a constant sentinel value that is compared when creating the entities from configuration.

7.210.3.2 DDS_ENTITY_NAME_USE_XML_CONFIG

```
const char* DDS_ENTITY_NAME_USE_XML_CONFIG  [extern]
```

<<***extension***>> *(*p. *432)* Special value to be used with **DDS_DomainParticipantConfigParams_t** (p. 964) to indicate that a participant created is with an autogenerated entity name.

This variable contains a constant sentinel value that is lexicographically compared when creating the entities from configuration.

7.210.3.3 DDS_QOS_ELEMENT_NAME_USE_XML_CONFIG

```
const char* DDS_QOS_ELEMENT_NAME_USE_XML_CONFIG  [extern]
```

<<***extension***>> *(p. 432)* Special value to be used with **DDS_DomainParticipantConfigParams_t** (p. 964) to indicate that entities are created from the QoS profile specified in the participant configuration.

This variable contains a constant sentinel value that is lexicographically compared when creating the entities from configuration.

7.211 User-managed Threads

User-managed thread infrastructure.

Classes

- class **DDSThreadFactory**

  <<***extension***>> (p. *432)* <<***interface***>> *(p. 433) Interface for providing the threads needed by the middleware. It consists of operations to create and delete threads. The interface depends on the* ***DDSThreadFactory_OnSpawned↩ Function*** *(p. 718) that specifies the operation to run in the new thread.*

Typedefs

- typedef DDS_ThreadFactory_OnSpawnedFunction **DDSThreadFactory_OnSpawnedFunction**

  Prototype of the function that must be called on a new thread created by a ***DDSThreadFactory*** *(p. 1883).*

7.211.1 Detailed Description

User-managed thread infrastructure.

Core feature that allows users to provide the threads to RTI Connext.

"::DDSThreadFactory"

The model follows the abstract factory pattern. A **DDSThreadFactory** (p. 1883) instance can be set in the **DDSDomainParticipantFactory** (p. 1694) so that **DDSDomainParticipant** (p. 1619) will use it for thread creation and deletion.

7.211.2 Typedef Documentation

7.211.2.1 DDSThreadFactory_OnSpawnedFunction

```
typedef DDS_ThreadFactory_OnSpawnedFunction  DDSThreadFactory_OnSpawnedFunction
```

Prototype of the function that must be called on a new thread created by a **DDSThreadFactory** (p. 1883).

Parameters

thread_param	<<***in***>> *(*p. *433)* Single argument that this operation receives. It is the data this operation needs to perform.

7.212 Monitoring Library 2.0

RTI Monitoring Library 2.0 public API.

Functions

- void **RTI_Monitoring_initialize** (void)

  Initializes Monitoring Library 2.0.

Variables

- const char ∗const **RTI_MONITORING_PERIODIC_TOPIC_NAME**

  The name of the Monitoring Topic used for periodic metrics distribution.
- const char ∗const **RTI_MONITORING_EVENT_TOPIC_NAME**

  The name of the Monitoring Topic used for event metrics distribution.
- const char ∗const **RTI_MONITORING_LOGGING_TOPIC_NAME**

  The name of the Monitoring Topic used for log messages distribution.

7.212.1 Detailed Description

RTI Monitoring Library 2.0 public API.

RTI Monitoring Library 2.0 is one component of the RTI Connext Observability Framework which allows collecting and distributing telemetry data (metrics and logs) associated with the observable resources created by an RTI Connext application.

In this release, the only Observable resources are the following entities: **DDSDataWriter** (p. 1589), **DDSDataReader** (p. 1556), **DDSPublisher** (p. 1818), **DDSSubscriber** (p. 1860), **DDSDomainParticipant** (p. 1619), **DDSTopic** (p. 1885) and Application (a process running RTI Connext).

The library also accepts remote commands to change the set of distributed telemetry data at run-time.

The data distributed by RTI Monitoring Library 2.0 is sent to an RTI Observability Collector Service instance, which forwards the data to other RTI Observability Collector Service instances or stores the data in third-party observability backends such as Prometheus or Grafana Loki.

RTI Monitoring Library 2.0 is a separate library (rtimonitoring2), and applications can use it in three different modes:

- **Dynamically loaded:** This is the default mode, and it requires that the rtimonitoring2 shared library is in the library search path.
- **Dynamic linking:** The application is linked with the rtimonitoring2 shared library.
- **Static linking:** The application is linked with the rtimonitoring2 static library.

The last two modes require calling the API **RTI_Monitoring_initialize** (p. 720) in your application before any other RTI Connext APIs. This API is defined in the header file `ndds/monitoring/monitoring_monitoringClass.h`.

Dynamic and static linking are only supported in C and C++ applications.

To enable use of RTI Monitoring Library 2.0 and configure its behavior, use the **DDS_MonitoringQosPolicy** (p. 1210) QoS policy on the **DDSDomainParticipantFactory** (p. 1694). This QoS policy can be configured programmatically or via XML.

7.212.2 Function Documentation

7.212.2.1 RTI_Monitoring_initialize()

```
void RTI_Monitoring_initialize (
             void  )
```

Initializes Monitoring Library 2.0.

This method must be called before calling any other RTI Connext API if the application links with RTI Monitoring Library 2.0 dynamically or statically.

7.212.3 Variable Documentation

7.212.3.1 RTI_MONITORING_PERIODIC_TOPIC_NAME

```
const char* const RTI_MONITORING_PERIODIC_TOPIC_NAME
```

The name of the Monitoring Topic used for periodic metrics distribution.

String version: "DCPSPeriodicStatusMonitoring".

This topic is used internally by Monitoring Library 2.0 and RTI Observability Collector Service. You should not publish or subscribe to it.

7.212.3.2 RTI_MONITORING_EVENT_TOPIC_NAME

```
const char* const RTI_MONITORING_EVENT_TOPIC_NAME
```

The name of the Monitoring Topic used for event metrics distribution.

String version: "DCPSEventStatusMonitoring".

This topic is used internally by Monitoring Library 2.0 and RTI Observability Collector Service. You should not publish or subscribe to it.

7.212.3.3 RTI_MONITORING_LOGGING_TOPIC_NAME

```
const char* const RTI_MONITORING_LOGGING_TOPIC_NAME
```

The name of the Monitoring Topic used for log messages distribution.

String version: "DCPSLoggingStatusMonitoring".

This topic is used internally by Monitoring Library 2.0 and RTI Observability Collector Service. You should not publish or subscribe to it.

7.213 Version

Retrieve information for the RTI Connext product, the core library, and the C, C++ or Java libraries.

Classes

- struct **NDDS_Config_LibraryVersion_t**
  The version of a single library shipped as part of an RTI Connext distribution.
- class **NDDSConfigVersion**
  *<<**interface**>> (p. 433) The version of an RTI Connext distribution.*

7.213.1 Detailed Description

Retrieve information for the RTI Connext product, the core library, and the C, C++ or Java libraries.

There are three ways to obtain version information: looking at the revision files, using Visual Studio or the command line, or programmatically at run time. This HTML documentation includes a reference for consulting the APIs that allow you to get version information programmatically. For more information see the RTI Connext DDS Core Libraries User's Manual.

The version information includes four fields:

- Major product version.
- Minor product version.
- Release letter for product version.
- Revision number of product.

7.214 Logging

Configure how much debugging information is reported during runtime and where it is logged.

Modules

- **Activity Context**

  Add contextual information to log messages.

Classes

- struct **NDDS_Config_LogMessage**

  A log message's input parameter to a logging device, including the text and additional information.
- class **NDDSConfigLoggerDevice**

  *<<**interface**>> (*p. *433) Logging device interface. Use for user-defined logging devices.*
- class **NDDSConfigLogger**

  *<<**interface**>> (*p. *433) The singleton type used to configure RTI Connext logging.*

Enumerations

- enum **NDDS_Config_LogVerbosity** {
  NDDS_CONFIG_LOG_VERBOSITY_SILENT ,
  NDDS_CONFIG_LOG_VERBOSITY_ERROR ,
  NDDS_CONFIG_LOG_VERBOSITY_WARNING ,
  NDDS_CONFIG_LOG_VERBOSITY_STATUS_LOCAL ,
  NDDS_CONFIG_LOG_VERBOSITY_STATUS_REMOTE ,
  NDDS_CONFIG_LOG_VERBOSITY_STATUS_ALL }

  The verbosities at which RTI Connext diagnostic information is logged.
- enum **NDDS_Config_LogLevel** {
  NDDS_CONFIG_LOG_LEVEL_FATAL_ERROR ,
  NDDS_CONFIG_LOG_LEVEL_ERROR ,
  NDDS_CONFIG_LOG_LEVEL_WARNING ,
  NDDS_CONFIG_LOG_LEVEL_STATUS_LOCAL ,
  NDDS_CONFIG_LOG_LEVEL_STATUS_REMOTE ,
  NDDS_CONFIG_LOG_LEVEL_DEBUG }

  Level category assigned to RTI Connext log messages returned to an output device.
- enum **NDDS_Config_SyslogLevel** {
  NDDS_CONFIG_SYSLOG_LEVEL_EMERGENCY ,
  NDDS_CONFIG_SYSLOG_LEVEL_ALERT ,
  NDDS_CONFIG_SYSLOG_LEVEL_CRITICAL ,
  NDDS_CONFIG_SYSLOG_LEVEL_ERROR ,
  NDDS_CONFIG_SYSLOG_LEVEL_WARNING ,
  NDDS_CONFIG_SYSLOG_LEVEL_NOTICE ,
  NDDS_CONFIG_SYSLOG_LEVEL_INFORMATIONAL ,
  NDDS_CONFIG_SYSLOG_LEVEL_DEBUG }

  Syslog level category assigned to RTI Connext log messages. See `Syslog Level and Verbosity Mapping`*, in the Core Libraries User's Manual, for more information.*

- enum **NDDS_Config_LogCategory** {
  NDDS_CONFIG_LOG_CATEGORY_PLATFORM ,
  NDDS_CONFIG_LOG_CATEGORY_COMMUNICATION ,
  NDDS_CONFIG_LOG_CATEGORY_DATABASE ,
  NDDS_CONFIG_LOG_CATEGORY_ENTITIES ,
  NDDS_CONFIG_LOG_CATEGORY_API ,
  NDDS_CONFIG_LOG_CATEGORY_DISCOVERY ,
  NDDS_CONFIG_LOG_CATEGORY_SECURITY ,
  NDDS_CONFIG_LOG_CATEGORY_USER ,
  NDDS_CONFIG_LOG_CATEGORY_ALL }

  Categories of logged messages.
- enum **NDDS_Config_LogPrintFormat** { }

  The format used to output RTI Connext diagnostic information.
- enum **NDDS_Config_LogFacility** {
  NDDS_CONFIG_LOG_FACILITY_USER ,
  NDDS_CONFIG_LOG_FACILITY_SECURITY_EVENT ,
  NDDS_CONFIG_LOG_FACILITY_SERVICE ,
  NDDS_CONFIG_LOG_FACILITY_MIDDLEWARE }

  A number that identifies the source of a log message.
- enum **NDDS_Config_SyslogVerbosity** {
  NDDS_CONFIG_SYSLOG_VERBOSITY_SILENT ,
  NDDS_CONFIG_SYSLOG_VERBOSITY_EMERGENCY ,
  NDDS_CONFIG_SYSLOG_VERBOSITY_ALERT ,
  NDDS_CONFIG_SYSLOG_VERBOSITY_CRITICAL ,
  NDDS_CONFIG_SYSLOG_VERBOSITY_ERROR ,
  NDDS_CONFIG_SYSLOG_VERBOSITY_WARNING ,
  NDDS_CONFIG_SYSLOG_VERBOSITY_NOTICE ,
  NDDS_CONFIG_SYSLOG_VERBOSITY_INFORMATIONAL ,
  NDDS_CONFIG_SYSLOG_VERBOSITY_DEBUG }

  The Syslog verbosities at which RTI Connext diagnostic information is logged. These Syslog verbosities are mapped to ***NDDS_Config_LogVerbosity*** *(*p. 723*). See* `Syslog Level and Verbosity Mapping`*, in the Core Libraries User's Manual, for more information.*

7.214.1 Detailed Description

Configure how much debugging information is reported during runtime and where it is logged.

7.214.2 Enumeration Type Documentation

7.214.2.1 NDDS_Config_LogVerbosity

`enum` **`NDDS_Config_LogVerbosity`**

The verbosities at which RTI Connext diagnostic information is logged.

Enumerator

NDDS_CONFIG_LOG_VERBOSITY_SILENT	No further output will be logged.
NDDS_CONFIG_LOG_VERBOSITY_ERROR	Only error and fatal error messages will be logged. An error indicates something wrong in the functioning of RTI Connext. The most common cause of errors is incorrect configuration.
NDDS_CONFIG_LOG_VERBOSITY_WARNING	Both error and warning messages will be logged. A warning indicates that RTI Connext is taking an action that may or may not be what you intended. Some configuration information is also logged at this verbosity to aid in debugging.
NDDS_CONFIG_LOG_VERBOSITY_STATUS_LOCAL	Errors, warnings, and verbose information about the lifecycles of local RTI Connext objects will be logged.
NDDS_CONFIG_LOG_VERBOSITY_STATUS_↩ REMOTE	Errors, warnings, and verbose information about the lifecycles of remote RTI Connext objects will be logged.
NDDS_CONFIG_LOG_VERBOSITY_STATUS_ALL	Errors, warnings, verbose information about the lifecycles of local and remote RTI Connext objects, and periodic information about RTI Connext threads will be logged.

7.214.2.2 NDDS_Config_LogLevel

enum **NDDS_Config_LogLevel**

Level category assigned to RTI Connext log messages returned to an output device.

Enumerator

NDDS_CONFIG_LOG_LEVEL_FATAL_ERROR	The message describes a fatal error. A fatal error indicates an unrecoverable situation in the functioning of RTI Connext. Error messages with this log level usually indicate a violation of an internal invariant or a segfault, and may include the function call stack where the fatal error happened.
NDDS_CONFIG_LOG_LEVEL_ERROR	The message describes an error. An error indicates a non-fatal problem in the functioning of RTI Connext. Errors are usually recoverable and will not stop application execution, although they may prevent some features from working properly. The most common cause of non-fatal errors is incorrect configuration and incorrect arguments.
NDDS_CONFIG_LOG_LEVEL_WARNING	The message describes a warning. A warning indicates that RTI Connext is taking an action that may or may not be what you intended. Some configuration information is also logged at this verbosity to aid in debugging.
NDDS_CONFIG_LOG_LEVEL_STATUS_LOCAL	The message contains information about the lifecycles of local RTI Connext objects will be logged.
NDDS_CONFIG_LOG_LEVEL_STATUS_REMOTE	The message contains information about the lifecycles of remote RTI Connext objects will be logged.
NDDS_CONFIG_LOG_LEVEL_DEBUG	The message contains debug information that might be relevant to your application.

7.214.2.3 NDDS_Config_SyslogLevel

enum **NDDS_Config_SyslogLevel**

Syslog level category assigned to RTI Connext log messages. See `Syslog Level and Verbosity Mapping`, in the Core Libraries User's Manual, for more information.

Enumerator

NDDS_CONFIG_SYSLOG_LEVEL_EMERGENCY	System is unusable. Maps to **NDDS_CONFIG_LOG_LEVEL_FATAL_ERROR** (p. 724).
NDDS_CONFIG_SYSLOG_LEVEL_ALERT	Should be corrected immediately. Maps to **NDDS_CONFIG_LOG_LEVEL_ERROR** (p. 724).
NDDS_CONFIG_SYSLOG_LEVEL_CRITICAL	Critical conditions. Maps to **NDDS_CONFIG_LOG_LEVEL_ERROR** (p. 724).
NDDS_CONFIG_SYSLOG_LEVEL_ERROR	Error conditions. Maps to **NDDS_CONFIG_LOG_LEVEL_ERROR** (p. 724).
NDDS_CONFIG_SYSLOG_LEVEL_WARNING	May indicate that an error will occur if action is not taken. Maps to **NDDS_CONFIG_LOG_LEVEL_WARNING** (p. 724).
NDDS_CONFIG_SYSLOG_LEVEL_NOTICE	Events that are unusual, but not error conditions. Maps to **NDDS_CONFIG_LOG_LEVEL_STATUS_LOCAL** (p. 724).
NDDS_CONFIG_SYSLOG_LEVEL_INFORMATIONAL	Normal operational messages that require no action. Maps to **NDDS_CONFIG_LOG_LEVEL_STATUS_LOCAL** (p. 724).
NDDS_CONFIG_SYSLOG_LEVEL_DEBUG	Information useful to developers for debugging the application. Maps to **NDDS_CONFIG_LOG_LEVEL_DEBUG** (p. 724).

7.214.2.4 NDDS_Config_LogCategory

enum **NDDS_Config_LogCategory**

Categories of logged messages.

The **NDDSConfigLogger::get_verbosity_by_category** (p. 2095) and **NDDSConfigLogger::set_verbosity_by_↩ category** (p. 2095) can be used to specify different verbosities for different categories of messages.

Enumerator

NDDS_CONFIG_LOG_CATEGORY_PLATFORM	Log messages pertaining to the underlying platform (hardware and OS) on which RTI Connext is running are in this category.
NDDS_CONFIG_LOG_CATEGORY_↩ COMMUNICATION	Log messages pertaining to data serialization and deserialization and network traffic are in this category.
NDDS_CONFIG_LOG_CATEGORY_DATABASE	Log messages pertaining to the internal database in which RTI Connext objects are stored are in this category.
NDDS_CONFIG_LOG_CATEGORY_ENTITIES	Log messages pertaining to local and remote entities, and to a subset of the discovery process, are in this category. (To see all discovery-related messages, use the DISCOVERY category.)
NDDS_CONFIG_LOG_CATEGORY_API	Log messages pertaining to the API layer of RTI Connext (such as method argument validation) are in this category.
NDDS_CONFIG_LOG_CATEGORY_DISCOVERY	Log messages pertaining to discovery are in this category.
NDDS_CONFIG_LOG_CATEGORY_SECURITY	Log messages pertaining to security are in this category. These messages include any messages logged by RTI Connext components even if they are not related to the Security Plugins.
NDDS_CONFIG_LOG_CATEGORY_USER	Log messages that are generated by the user using the following log APIs: **NDDSConfigLogger::emergency** (p. 2098), **NDDSConfigLogger::alert** (p. 2098), **NDDSConfigLogger::critical** (p. 2099), **NDDSConfigLogger::error** (p. 2099), **NDDSConfigLogger::warning** (p. 2099), **NDDSConfigLogger::notice** (p. 2099), **NDDSConfigLogger::informational** (p. 2099), **NDDSConfigLogger::debug** (p. 2100).
NDDS_CONFIG_LOG_CATEGORY_ALL	Log messages pertaining to all categories in RTI Connext.

7.214.2.5 NDDS_Config_LogPrintFormat

enum **NDDS_Config_LogPrintFormat**

The format used to output RTI Connext diagnostic information.

Enumerator

NDDS_CONFIG_LOG_PRINT_FORMAT_DEFAULT	(default) Print message, method name, log level, **Activity Context** (p. 729) (what was happening when the event occurred), and logging category.
NDDS_CONFIG_LOG_PRINT_FORMAT_↩ TIMESTAMPED	Print message, method name, log level, **Activity Context** (p. 729), logging category, and timestamp.

Enumerator

NDDS_CONFIG_LOG_PRINT_FORMAT_VERBOSE	Print message with all available context information (includes thread identifier, message location).
NDDS_CONFIG_LOG_PRINT_FORMAT_VERBOSE↩ _TIMESTAMPED	Print message with all available context information, and timestamp.
NDDS_CONFIG_LOG_PRINT_FORMAT_DEBUG	Print a set of fields (including message number and backtrace information) that may be useful for internal debugging.
NDDS_CONFIG_LOG_PRINT_FORMAT_MINIMAL	Print only message number and message location.
NDDS_CONFIG_LOG_PRINT_FORMAT_MAXIMAL	Print all available fields.

7.214.2.6 NDDS_Config_LogFacility

enum **NDDS_Config_LogFacility**

A number that identifies the source of a log message.

In the Syslog Protocol, the Facility is a numerical code that represents the machine process that created a Syslog event. RTI Connext uses the facility to represent the source of a given log message.

Enumerator

NDDS_CONFIG_LOG_FACILITY_USER	A log message produced by the APIs that log user messages in **NDDSConfigLogger** (p. 2093) (e.g, **NDDSConfigLogger::emergency** (p. 2098)). Numerical code: 1
NDDS_CONFIG_LOG_FACILITY_SECURITY_EVENT	A security-related message logged by the RTI Security Plugins Logging Plugin. Numerical code: 10
NDDS_CONFIG_LOG_FACILITY_SERVICE	A log message produced by an Infrastructure Service, such as Routing Service. Infrastructure Services operate using the Core Libraries. The messages that the Core Libraries create are categorized under the MIDDLEWARE facility. Log messages that are directly generated by the Infrastructure Service itself are marked with the SERVICE facility. Numerical code: 22
NDDS_CONFIG_LOG_FACILITY_MIDDLEWARE	A log message produced by RTI Connext Core Libraries. Numerical code: 23

7.214.2.7 NDDS_Config_SyslogVerbosity

enum **NDDS_Config_SyslogVerbosity**

The Syslog verbosities at which RTI Connext diagnostic information is logged. These Syslog verbosities are mapped to **NDDS_Config_LogVerbosity** (p. 723). See `Syslog Level and Verbosity Mapping`, in the Core Libraries User's Manual, for more information.

Enumerator

NDDS_CONFIG_SYSLOG_VERBOSITY_SILENT	No messages will be logged. (lowest verbosity) Equivalent to **NDDS_CONFIG_LOG_VERBOSITY_SILENT** (p. 724).
NDDS_CONFIG_SYSLOG_VERBOSITY_↩ EMERGENCY	Emergency, critical, alert, and error messages will be logged. This Syslog verbosity level is translated to **NDDS_CONFIG_LOG_VERBOSITY_ERROR** (p. 724) when interpreted by RTI Connext. (That's why this level actually logs more than just emergency messages.)
NDDS_CONFIG_SYSLOG_VERBOSITY_ALERT	Emergency, critical, and alert messages will be logged. This Syslog verbosity level is translated to **NDDS_CONFIG_LOG_VERBOSITY_ERROR** (p. 724) when interpreted by RTI Connext. (That's why this level actually logs more than just emergency, critical and alert messages.)
NDDS_CONFIG_SYSLOG_VERBOSITY_CRITICAL	Emergency, critical, alert, and error messages will be logged. This Syslog verbosity level is translated to **NDDS_CONFIG_LOG_VERBOSITY_ERROR** (p. 724) when interpreted by RTI Connext. (That's why this level actually logs more than just emergency and critical messages.)
NDDS_CONFIG_SYSLOG_VERBOSITY_ERROR	Emergency, critical, alert, and error messages will be logged. This Syslog verbosity level is translated to **NDDS_CONFIG_LOG_VERBOSITY_ERROR** (p. 724) when interpreted by RTI Connext.
NDDS_CONFIG_SYSLOG_VERBOSITY_WARNING	Emergency, critical, alert, error, and warning messages will be logged. This Syslog verbosity level is translated to **NDDS_CONFIG_LOG_VERBOSITY_WARNING** (p. 724) when interpreted by RTI Connext.
NDDS_CONFIG_SYSLOG_VERBOSITY_NOTICE	Emergency, critical, alert, error, warning, notice and informational messages will be logged. This Syslog verbosity level is translated to **NDDS_CONFIG_LOG_↩ VERBOSITY_STATUS_REMOTE** (p. 724) when interpreted by RTI Connext. (That's why this level actually logs more than just emergency, critical, alert, error, warning and notice messages.)
NDDS_CONFIG_SYSLOG_VERBOSITY_↩ INFORMATIONAL	Emergency, critical, alert, error, warning, notice, and informational messages will be logged. This Syslog verbosity level is translated to **NDDS_CONFIG_LOG_↩ VERBOSITY_STATUS_REMOTE** (p. 724) when interpreted by RTI Connext.
NDDS_CONFIG_SYSLOG_VERBOSITY_DEBUG	All messages will be logged. This Syslog verbosity level is translated to **NDDS_CONFIG_LOG_VERBOSITY_STATUS_ALL** (p. 724) when interpreted by RTI Connext.

7.215 Activity Context

Add contextual information to log messages.

Classes

- class **NDDSConfigActivityContext**

  Activity Context APIs.

Macros

- #define **NDDS_CONFIG_ACTIVITY_CONTEXT_ATTRIBUTE_MASK_DEFAULT** RTI_OSAPI_ACTIVITY_↩ CONTEXT_ATTRIBUTE_MASK_DEFAULT

  Provide the default attributes of the resource of the Activity Context.
- #define **NDDS_CONFIG_ACTIVITY_CONTEXT_ATTRIBUTE_MASK_NONE** RTI_OSAPI_ACTIVITY_↩ CONTEXT_ATTRIBUTE_MASK_NONE

  Not provide any attribute of the resource of the Activity Context.
- #define **NDDS_CONFIG_ACTIVITY_CONTEXT_ATTRIBUTE_MASK_ALL** RTI_OSAPI_ACTIVITY_↩ CONTEXT_ATTRIBUTE_MASK_ALL

  Provide all the possibles attributes of the resource of the Activity Context.

Typedefs

- typedef **DDS_Long NDDS_Config_ActivityContextAttributeKindMask**

  *The attributes **NDDS_Config_ActivityContextAttributeKind** (p. 731) used in the string representation of the Activity Context can be configured through this mask.*

Enumerations

- enum **NDDS_Config_ActivityContextAttributeKind** {
  NDDS_CONFIG_ACTIVITY_CONTEXT_ATTRIBUTE_GUID_PREFIX ,
  NDDS_CONFIG_ACTIVITY_CONTEXT_ATTRIBUTE_TOPIC ,
  NDDS_CONFIG_ACTIVITY_CONTEXT_ATTRIBUTE_TYPE ,
  NDDS_CONFIG_ACTIVITY_CONTEXT_ATTRIBUTE_ENTITY_KIND ,
  NDDS_CONFIG_ACTIVITY_CONTEXT_ATTRIBUTE_DOMAIN_ID ,
  NDDS_CONFIG_ACTIVITY_CONTEXT_ATTRIBUTE_ENTITY_NAME }

  *The resources of the **Activity Context** (p. 729) can have multiple associated attributes. Those attributes provide extra information about the entity such as GUID prefix, Topic, data type, entity kind, entity name and domain ID. They are used to indicate what attributes of the resources are included in the activity context.*

Functions

- static void **NDDSConfigActivityContext::set_attribute_mask** (**NDDS_Config_ActivityContextAttribute**↩ **KindMask** attribute_format)

  *Set the **NDDS_Config_ActivityContextAttributeKindMask** (p. 731) of the Activity Context.*

7.215.1 Detailed Description

Add contextual information to log messages.

The Activity Context is a group of resources and activities associated with an action such as the creation of an entity.

- A resource is a abstraction of an entity. It can contain attributes such as Topic or domain ID.
- An activity is a general task that the resource is doing, such as "Getting QoS".

Logging context is one of the formats RTI Connext logging infrastructure supports. It is used by default in **NDDS_CONFIG_LOG_PRINT_FORMAT_DEFAULT** (p. 726). It provides information about resources and activities. The activity context is used in two places:

- Logging: activity context is one of the **NDDS_Config_LogPrintFormat** (p. 726) DDS logging infrastructure supports. If that format is set every time RTI Connext logs a message, it will contain the contextual information.
- Heap monitoring: every time memory is allocated and heap monitoring is enabled, the string representation of the activity context will be associated with the allocation. This information will be available when taking the snapshot.

For example, in the creation of a DataWriter, the activity context will provide information about:

- Resource: the Publisher creating the DataWriter. Attributes of the publisher will be GUID, kind, name and domain ID.
- Activity: entity creation. It will have two parameters, the entity kind and the Topic. In the example below, these are "Writer" and "TestTopic".

The string representation of the above activity context would be:

```
[0X101A76B,0X79E5D71,0X50EE914:0X1C1:0X80000088{Entity=Pu,Name=TestPublisher,Domain=1}|CREATE Writer WITH TOPIC
       TestTopic]
```

Another example could be when a DataWriter writes a sample. The activity context will provide information about:

- Resource: the DataWriter writing the sample. The attributes of the DataWriter will be GUID, name, kind, Topic, data type, and the domain ID.
- Activity will be "write a sample".

The string representation of the activity context will be:

```
[0X101A76B,0X79E5D71,0X50EE914:0X1C1:0X80000003{Name=testDataWriterName,Entity=DW,Topic=test,Type=Foo,Domain=1}|Write]
```

7.215.2 Macro Definition Documentation

7.215.2.1 NDDS_CONFIG_ACTIVITY_CONTEXT_ATTRIBUTE_MASK_DEFAULT

```
#define NDDS_CONFIG_ACTIVITY_CONTEXT_ATTRIBUTE_MASK_DEFAULT  RTI_OSAPI_ACTIVITY_CONTEXT_ATTRIBUTE↩
_MASK_DEFAULT
```

Provide the default attributes of the resource of the Activity Context.

7.215.2.2 NDDS_CONFIG_ACTIVITY_CONTEXT_ATTRIBUTE_MASK_NONE

```
#define NDDS_CONFIG_ACTIVITY_CONTEXT_ATTRIBUTE_MASK_NONE  RTI_OSAPI_ACTIVITY_CONTEXT_ATTRIBUTE_↩
MASK_NONE
```

Not provide any attribute of the resource of the Activity Context.

7.215.2.3 NDDS_CONFIG_ACTIVITY_CONTEXT_ATTRIBUTE_MASK_ALL

```
#define NDDS_CONFIG_ACTIVITY_CONTEXT_ATTRIBUTE_MASK_ALL  RTI_OSAPI_ACTIVITY_CONTEXT_ATTRIBUTE_↩
MASK_ALL
```

Provide all the possibles attributes of the resource of the Activity Context.

7.215.3 Typedef Documentation

7.215.3.1 NDDS_Config_ActivityContextAttributeKindMask

```
typedef DDS_Long NDDS_Config_ActivityContextAttributeKindMask
```

The attributes **NDDS_Config_ActivityContextAttributeKind** (p. 731) used in the string representation of the Activity Context can be configured through this mask.

This mask indicates what attributes of the resource are used when RTI Connext logs a message or when the Heap Monitoring utility saves statistics for a memory allocation.

7.215.4 Enumeration Type Documentation

7.215.4.1 NDDS_Config_ActivityContextAttributeKind

```
enum NDDS_Config_ActivityContextAttributeKind
```

The resources of the **Activity Context** (p. 729) can have multiple associated attributes. Those attributes provide extra information about the entity such as GUID prefix, Topic, data type, entity kind, entity name and domain ID. They are used to indicate what attributes of the resources are included in the activity context.

Enumerator

NDDS_CONFIG_ACTIVITY_CONTEXT_↩ ATTRIBUTE_GUID_PREFIX	Provide the entity GUID prefix to the resource of the Activity Context. For example:
	• For the following string representation of the context:
	`[0X101A76B,0X79E5D71,0X50EE914:0X1C1:0X80000003{Name=testDataWrite`
	• The GUID prefix is 0X101A76B,0X79E5D71,0X50EE914. If the bit **NDDS_CONFIG_ACTIVITY_CONTEXT_↩ ATTRIBUTE_GUID_PREFIX** (p. 732) is not set, the string representation will not show the GUID prefix:
	`[:0X80000003{Name=testDataWriterName,Entity=DW,Topic=test,Type=Foo`

NDDS_CONFIG_ACTIVITY_CONTEXT_↩ ATTRIBUTE_TOPIC	Provide the Topic to the resource of the Activity Context. The topic attribute is specified by "Topic". For example: • For the following string representation of the context: `[0X101A76B,0X79E5D71,0X50EE914:0X1C1:0X80000003{Name=testDataWrite` • The Topic is "test." If the bit **NDDS_CONFIG_↩ ACTIVITY_CONTEXT_ATTRIBUTE_TOPIC** (p. 732) is not set, the string representation will not show the Topic: `[0X101A76B,0X79E5D71,0X50EE914:0X1C1:0X80000003{Name=testDataWrite`
NDDS_CONFIG_ACTIVITY_CONTEXT_↩ ATTRIBUTE_TYPE	Provide the data type to the resource of the Activity Context. The type attribute is specified by "Type". For example: • For the following string representation of the context: `[0X101A76B,0X79E5D71,0X50EE914:0X1C1:0X80000003{Name=testDataWrite` • The data type is "Foo." If the bit **NDDS_CONFIG_ACTIVITY_CONTEXT_↩ ATTRIBUTE_TYPE** (p. 732) is not set, the string representation will not show the data type: `[0X101A76B,0X79E5D71,0X50EE914:0X1C1:0X80000003{Name=testDataWrite`

Enumerator

NDDS_CONFIG_ACTIVITY_CONTEXT_↩ ATTRIBUTE_ENTITY_KIND	Provide the entity kind to the resource of the Activity Context. The kind attribute is specified by "Entity". For example: • For the following string representation of the context: `[0X101A76B,0X79E5D71,0X50EE914:0X1C1:0X80000003{Name=testDataWrite` • The entity kind is "Writer." If the bit **NDDS_CONFIG_ACTIVITY_CONTEXT_↩ ATTRIBUTE_ENTITY_KIND** (p. 733) is not set, the string representation will not show the entity kind: `[0X101A76B,0X79E5D71,0X50EE914:0X1C1:0X80000003{Name=testDataWrite`
NDDS_CONFIG_ACTIVITY_CONTEXT_↩ ATTRIBUTE_DOMAIN_ID	Provide the domain ID to the resource of the Activity Context. The domain attribute is specified by "Domain". For example: • For the following string representation of the context: `[0X101A76B,0X79E5D71,0X50EE914:0X1C1:0X80000003{Name=testDataWrite` • The domain ID is "1." If the bit **NDDS_CONFIG_ACTIVITY_CONTEXT_↩ ATTRIBUTE_DOMAIN_ID** (p. 733) is not set, the string representation will not show the domain ID: `[0X101A76B,0X79E5D71,0X50EE914:0X1C1:0X80000003{Name=testDataWrite`

NDDS_CONFIG_ACTIVITY_CONTEXT_↩ ATTRIBUTE_ENTITY_NAME	Provide the entity name to the resource of the Activity Context. The name attribute is specified by "Name". For example:
	• For the following string representation of the context:
	`[0X101A76B,0X79E5D71,0X50EE914:0X1C1:0X80000003{Name=testDataWrite`
	• The entity name is "testDataWriterName." If the bit **NDDS_CONFIG_ACTIVITY_CONTEXT_↩ ATTRIBUTE_ENTITY_NAME** (p. 733) is not set, the string representation will not show the entity name:
	`[0X101A76B,0X79E5D71,0X50EE914:0X1C1:0X80000003{Entity=DW,Topic=te`

7.215.5 Function Documentation

7.215.5.1 set_attribute_mask()

```
static void NDDSConfigActivityContext::set_attribute_mask (
              NDDS_Config_ActivityContextAttributeKindMask attribute_format )  [static]
```

Set the **NDDS_Config_ActivityContextAttributeKindMask** (p. 731) of the Activity Context.

7.216 Compliance Configuration

APIs to configure compliance with certain standard specifications.

Classes

- class **NDDSConfigCompliance**

  APIs to configure all the compliance mask.

Macros

- #define **NDDS_CONFIG_XTYPES_COMPLIANCE_MASK_VENDOR**

  XTypes vendor compliance mask. This value is fully aligned with the XTypes specification.
- #define **NDDS_CONFIG_XTYPES_COMPLIANCE_MASK_DEFAULT**

  Default XTypes compliance mask.
- #define **NDDS_CONFIG_RPC_COMPLIANCE_MASK_VENDOR**

  This mask configures compliance with the OMG DDS-RPC specification.
- #define **NDDS_CONFIG_RPC_COMPLIANCE_MASK_DEFAULT**

  Default RPC compliance mask.

Typedefs

- typedef **DDS_UnsignedLong NDDS_Config_XTypesComplianceMask**

  XTypes compliance mask.
- typedef **DDS_UnsignedLong NDDS_Config_RpcComplianceMask**

  Mask to configure compliance with the OMG DDS-RPC specification.

Enumerations

- enum **NDDS_Config_XTypesComplianceMaskBits** {
  NDDS_CONFIG_XTYPES_DHEADER_IN_NON_PRIMITIVE_COLLECTIONS_BIT ,
  NDDS_CONFIG_XTYPES_ENUM_AS_PRIMITIVE_IN_COLLECTIONS_BIT ,
  NDDS_CONFIG_XTYPES_PARAMETER_LENGTH_WITH_PADDING_BIT ,
  NDDS_CONFIG_XTYPES_ENCAPSULATION_OPTIONS_WITH_PADDING_BIT ,
  NDDS_CONFIG_XTYPES_ACCEPT_UNKNOWN_ENUM_VALUE_BIT ,
  NDDS_CONFIG_XTYPES_ACCEPT_UNKNOWN_DISCRIMINATOR_BIT ,
  NDDS_CONFIG_XTYPES_SELECT_DEFAULT_DISCRIMINATOR_BIT ,
  NDDS_CONFIG_XTYPES_SENTINEL_IN_EMPTY_UNION_BIT ,
  NDDS_CONFIG_XTYPES_INITIALIZE_DISCRIMINATOR_TO_DEFAULT_BIT ,
  NDDS_CONFIG_XTYPES_JAVA_KEYHASH_750_BIT ,
  NDDS_CONFIG_XTYPES_PYTHON_INT8_730_BIT ,
  NDDS_CONFIG_XTYPES_MUST_UNDERSTAND_IN_KEY_AND_DISCRIMINATOR_BIT }

  *The bits for the **NDDS_Config_XTypesComplianceMask** (p. 736) with different aspects of the Extended CDR encoding (XCDR and XCDR2).*
- enum **NDDS_Config_RpcComplianceMaskBits** { **NDDS_CONFIG_RPC_USE_STANDARD_RELATED_SAMPLE_PID_BIT** }

  *The bits for the **NDDS_Config_RpcComplianceMask** (p. 737) with different aspects of RPC and Request Reply.*

7.216.1 Detailed Description

APIs to configure compliance with certain standard specifications.

7.216.2 Macro Definition Documentation

7.216.2.1 NDDS_CONFIG_XTYPES_COMPLIANCE_MASK_VENDOR

```
#define NDDS_CONFIG_XTYPES_COMPLIANCE_MASK_VENDOR
```

XTypes vendor compliance mask. This value is fully aligned with the XTypes specification.

The mask sets the following bits:

- **NDDS_CONFIG_XTYPES_DHEADER_IN_NON_PRIMITIVE_COLLECTIONS_BIT** (p. 739)
- **NDDS_CONFIG_XTYPES_ENCAPSULATION_OPTIONS_WITH_PADDING_BIT** (p. 740)
- **NDDS_CONFIG_XTYPES_ACCEPT_UNKNOWN_DISCRIMINATOR_BIT** (p. 741)
- **NDDS_CONFIG_XTYPES_SENTINEL_IN_EMPTY_UNION_BIT** (p. 742)
- **NDDS_CONFIG_XTYPES_INITIALIZE_DISCRIMINATOR_TO_DEFAULT_BIT** (p. 743)

The HEX value for the mask is: 0x000001a9.

Warning

This mask will always include all known fixes to ensure full alignment with the XTypes specification. As a result, its contents may change between releases or patches. If you want to ensure that your application's behavior remains unaffected by future changes, you should specify the exact bits you are interested in.

7.216.2.2 NDDS_CONFIG_XTYPES_COMPLIANCE_MASK_DEFAULT

```
#define NDDS_CONFIG_XTYPES_COMPLIANCE_MASK_DEFAULT
```

Default XTypes compliance mask.

The mask sets the following bits:

- **NDDS_CONFIG_XTYPES_PARAMETER_LENGTH_WITH_PADDING_BIT** (p. 739)
- **NDDS_CONFIG_XTYPES_ENCAPSULATION_OPTIONS_WITH_PADDING_BIT** (p. 740)
- **NDDS_CONFIG_XTYPES_SENTINEL_IN_EMPTY_UNION_BIT** (p. 742)
- **NDDS_CONFIG_XTYPES_INITIALIZE_DISCRIMINATOR_TO_DEFAULT_BIT** (p. 743)

The HEX value for the mask is: 0x0000018C.

7.216.2.3 NDDS_CONFIG_RPC_COMPLIANCE_MASK_VENDOR

```
#define NDDS_CONFIG_RPC_COMPLIANCE_MASK_VENDOR
```

This mask configures compliance with the OMG DDS-RPC specification.

The mask sets the following bits:

- **NDDS_CONFIG_RPC_USE_STANDARD_RELATED_SAMPLE_PID_BIT** (p. 745)

The HEX value for the mask is: 0x00000001.

7.216.2.4 NDDS_CONFIG_RPC_COMPLIANCE_MASK_DEFAULT

```
#define NDDS_CONFIG_RPC_COMPLIANCE_MASK_DEFAULT
```

Default RPC compliance mask.

The mask does not sets any bits.

The HEX value for the mask is: 0x00000000.

7.216.3 Typedef Documentation

7.216.3.1 NDDS_Config_XTypesComplianceMask

```
typedef DDS_UnsignedLong NDDS_Config_XTypesComplianceMask
```

XTypes compliance mask.

The **NDDS_Config_XTypesComplianceMask** (p. 736) configures different aspects of the Extended CDR encoding (XCDR and XCDR2).

Applications that don't use the exact same XTypes compliance mask may not be interoperable.

By default, data serialization is not fully compliant with Extended CDR encoding. While this doesn't affect correctness, it may prevent interoperability with other vendors.

You can change the default compliance mask for an application using any of the following mechanisms:

- Setting the environment variable **NDDS_XTYPES_COMPLIANCE_MASK**
- Calling the function **NDDSConfigCompliance::set_xtypes_mask** (p. 2090)

The environment variable will be automatically loaded when a **DDSDomainParticipantFactory** (p. 1694) is created. If you need to load it before that, use the function **NDDSConfigCompliance::load_compliance_masks** (p. 2091).

The format of the environment variable is an unsigned integer, which can be in HEX notation. For example: 0x00000001.

The default value is **NDDS_CONFIG_XTYPES_COMPLIANCE_MASK_DEFAULT** (p. 735).

To be fully compliant with the XTypes specification, set the mask to **NDDS_CONFIG_XTYPES_COMPLIANCE_↩ MASK_VENDOR** (p. 735).

[default] NDDS_CONFIG_XTYPES_COMPLIANCE_MASK_DEFAULT (p. 735)

When changing the mask for an application is not desired, the property dds.xtypes.compliance_mask will change the mask at the DomainParticipant or DataWriter level. When set in the DomainParticipant, it configures the mask for all DataWriters in the participant. See `Extensible Types Compliance Mask` in the RTI Connext Core Libraries Extensible Types Guide.

7.216.3.2 NDDS_Config_RpcComplianceMask

```
typedef DDS_UnsignedLong NDDS_Config_RpcComplianceMask
```

Mask to configure compliance with the OMG DDS-RPC specification.

The **NDDS_Config_RpcComplianceMask** (p. 737) configures different aspects of RPC communication.

By default, the PID_RELATED_SAMPLE_IDENTITY used in RPC is not compliant with DDS RPC specification. This issue only affect interoperability with other vendors.

You can change the default compliance mask for an application using any of the following mechanisms:

- Setting the environment variable **NDDS_RPC_COMPLIANCE_MASK**

- Calling the function **NDDSConfigCompliance::set_rpc_mask** (p. 2092)

The environment variable will be automatically loaded when a **DDSDomainParticipantFactory** (p. 1694) is created. If you need to load it before that, use the function **NDDSConfigCompliance::load_compliance_masks** (p. 2091).

The format of the environment variable is an unsigned integer, which can be in HEX notation. For example: 0x00000001.

The default value is **NDDS_CONFIG_RPC_COMPLIANCE_MASK_DEFAULT** (p. 736).

To be fully compliant with the DDS RPC specification, set the mask to **NDDS_CONFIG_RPC_COMPLIANCE_MASK↩ _VENDOR** (p. 736).

[default] NDDS_CONFIG_RPC_COMPLIANCE_MASK_DEFAULT (p. 736)

If changing the the mask in an application is not desired, the property dds.rpc.compliance_mask will change the mask at Participant or DataWriter level. When set in the Participant it configures the mask for all DataWriters in the Participant. See Using RPC compliance mask in the RTI Connext Core Libraries Users Manual.

7.216.4 Enumeration Type Documentation

7.216.4.1 NDDS_Config_XTypesComplianceMaskBits

`enum` **`NDDS_Config_XTypesComplianceMaskBits`**

The bits for the **NDDS_Config_XTypesComplianceMask** (p. 736) with different aspects of the Extended CDR encoding (XCDR and XCDR2).

See also

NDDS_Config_XTypesComplianceMask (p. 736)

Enumerator

NDDS_CONFIG_XTYPES_DHEADER_IN_NON_↩ PRIMITIVE_COLLECTIONS_BIT	When this bit is set, the serialization of sequences and arrays with non-primitive members includes a DHEADER. Value: 0x00000001 To be compatible with the XTypes specification, this bit must be set. This bit only applies to XCDR2 encoding. Limitation: For FlatData, the compliant encoding is applied to arrays and sequences of mutable types, but not final types . The serialization of final types containing arrays of non-primitive types in the FlatData language binding is not compliant with Extended CDR encoding version 2 in any case. This bit is unset by default. See **NDDS_CONFIG_↩ XTYPES_COMPLIANCE_MASK_DEFAULT** (p. 735)
NDDS_CONFIG_XTYPES_ENUM_AS_PRIMITIVE_↩ IN_COLLECTIONS_BIT	When this bit is set, enums are considered primitive types in collection types. Value: 0x00000002 A DHEADER will not be added to collections of enums if **NDDS_CONFIG_XTYPES_DHEADER_IN_NON_↩ PRIMITIVE_COLLECTIONS_BIT** (p. 739) is set. To be compatible with the XTypes specification, this bit must be unset. This bit only applies to XCDR2 encoding. This bit is unset by default. See **NDDS_CONFIG_↩ XTYPES_COMPLIANCE_MASK_DEFAULT** (p. 735)
NDDS_CONFIG_XTYPES_PARAMETER_LENGTH↩ _WITH_PADDING_BIT	When this bit is set, the length of a member header in a mutable type or an optional member will include padding bytes to a 4-byte boundary. Value: 0x00000004 To be compatible with the XTypes specification, this bit must be unset. This bit only applies to XCDR encoding. This bit is set by default. See **NDDS_CONFIG_↩ XTYPES_COMPLIANCE_MASK_DEFAULT** (p. 735)

Enumerator

NDDS_CONFIG_XTYPES_ENCAPSULATION_↩ OPTIONS_WITH_PADDING_BIT	When this bit is set, RTI Connext will set the padding bits in the options field of the encapsulation header of a serialized payload. Value: 0x00000008 When this bit is set RTI Connext will set the least significant two bits in the second byte of the options field to a value that encodes the number of padding bytes needed after the end of the serialized payload in order to reach the next 4-byte aligned offset. Specifically, the least significant two bits shall be set to binary 00 if no padding bits would be needed and binary 01, 10, or 11 if, respectively, there would be one, two, or three bytes of padding needed. These bits in the options field will be interpreted by a DataReader to determine where the serialized data exactly ended. To be compatible with the XTypes specification, this bit must be set. To be compatible with the following Micro versions this bit must be unset because those versions of Micro did not support padding bits and will discard any data with encapsulations options not set to 0. • RTI Connext Micro 3.x.y.z • RTI Connext Micro 2.4.15.1 • RTI Connext Micro 2.4.14.0 and 2.4.14.1 • RTI Connext Micro 2.4.13.1 through 2.4.13.5 • RTI Connext Micro 2.4.12.z This bit applies to both, XCDR and XCDR2 encodings. This bit is set by default. See **NDDS_CONFIG_↩ XTYPES_COMPLIANCE_MASK_DEFAULT** (p. 735)
NDDS_CONFIG_XTYPES_ACCEPT_UNKNOWN_↩ ENUM_VALUE_BIT	When this bit is set, samples containing an unknown enumerator can be successfully deserialized to the default enumeration value. Value: 0x00000010 To be compatible with the XTypes specification, this bit must be unset. This bit is unset by default. See **NDDS_CONFIG_↩ XTYPES_COMPLIANCE_MASK_DEFAULT** (p. 735)

Enumerator

NDDS_CONFIG_XTYPES_ACCEPT_UNKNOWN_↩ DISCRIMINATOR_BIT	When this bit is set, samples containing an unknown discriminator can be successfully deserialized, and the sample won't be dropped. Value: 0x00000020 When this bit is set, samples containing an unknown discriminator won't be dropped, and the discriminator value assigned to the sample will depend on whether or not the bit **NDDS_CONFIG_XTYPES_SELECT_↩ DEFAULT_DISCRIMINATOR_BIT** (p. 741) is set. To be compatible with the XTypes specification, this bit must be set. This bit is unset by default. See **NDDS_CONFIG_↩ XTYPES_COMPLIANCE_MASK_DEFAULT** (p. 735)
NDDS_CONFIG_XTYPES_SELECT_DEFAULT_↩ DISCRIMINATOR_BIT	When this bit is set, samples containing a discriminator value that does not select any member in the union will be assigned the default discriminator value; if the bit is not set, the sample will preserve the discriminator value. Value: 0x00000040 When this bit is set, samples containing a discriminator value that does not select any member in the union will be assigned the default discriminator value. If the default value selects a valid union member, that member will be initialized to the default value of its type. If the bit is not set, the sample will preserve the discriminator value. The bit **NDDS_CONFIG_XTYPES_INITIALIZE↩ _DISCRIMINATOR_TO_DEFAULT_BIT** (p. 743) determines what the default discriminator value is. This bit is only relevant when the bit **NDDS_CONFIG_XTYPES_ACCEPT_UNKNOWN_↩ DISCRIMINATOR_BIT** (p. 741) is set, and it is used to determine the value of the discriminator when the sample contains an unknown discriminator. To be compatible with the XTypes specification, this bit must be unset. This bit is unset by default. See **NDDS_CONFIG_↩ XTYPES_COMPLIANCE_MASK_DEFAULT** (p. 735)

Enumerator

NDDS_CONFIG_XTYPES_SENTINEL_IN_EMPTY_↩ UNION_BIT	When this bit is set, samples of a mutable union that contain an unknown discriminator will be serialized with a 4-byte sentinel value. Value: 0x00000080 When this bit is set, samples of a mutable union that contain an unknown discriminator will be serialized with a 4-byte sentinel value: 0x027f0000. This is the same sentinel value as the one used for samples of a mutable union that contain a known discriminator. This bit only applies to XCDR encoding, Dynamic Data, and language APIs other than Java. When using Java and not using Dynamic Data, the sentinel is always added. To be compatible with the XTypes specification and with Connext version 5.3.x, this bit must be set. This bit is set by default. See **NDDS_CONFIG_↩ XTYPES_COMPLIANCE_MASK_DEFAULT** (p. 735)

Enumerator

NDDS_CONFIG_XTYPES_INITIALIZE_↩ DISCRIMINATOR_TO_DEFAULT_BIT	When this bit is set, the union discriminator will be initialized to the default value of its type. Value: 0x00000100 When this bit is set, the union discriminator will be initialized to the default value of its type. This means 0 for numeric discriminators, false for boolean discriminators, and the first literal of the enumeration for enumerated discriminators. As a result, a union may be initialized with a discriminator value that does not correspond to any of its members. The default value of the type can be modified by creating an alias type and using the **[default]** annotation to specify the desired default value. You can then use that alias type as the discriminator type, and the default discriminator value will be the one specified in the alias type. When this bit is unset, the discriminator will be set to select to a value that selects the default union member, if one is defined, or otherwise to the lowest value associated with any member. If you generated code (using Code Generator) with this bit configured one way using the -xTypesComplianceMask flag (for example, -xTypesComplianceMask 0x00000100) and then configure it the opposite way in the API (for example, unset at 0x00000000), you must regenerate code using Code Generator's -xTypesComplianceMask flag configured the same way as it is in the API (for example, -xTypesComplianceMask 0x00000000). Note that this bit is the only bit that Code Generator uses; therefore, you have to set it the same both in the API and in Code Generator. See information about Code Generator's -xTypesComplianceMask command-line option in `Command-Line Arguments for rtiddsgen`, in the RTI Code Generator User's Manual. This bit affects **NDDS_CONFIG_XTYPES_SELECT_↩ DEFAULT_DISCRIMINATOR_BIT** (p. 741). (See that bit's description above.) To be compatible with the XTypes specification, this bit must be set. This bit is set by default. See **NDDS_CONFIG_↩ XTYPES_COMPLIANCE_MASK_DEFAULT** (p. 735)

Enumerator

NDDS_CONFIG_XTYPES_JAVA_KEYHASH_750_BIT	When this bit is set, the KeyHash calculation for a Java instance will match the one produced by the Java API in Connext 7.5.0 and earlier versions. Value: 0x00000200 When this bit is set, the KeyHash calculation for a Java instance will match the one produced by the Java API in Connext 7.5.0 and earlier versions. This affects KeyHash calculation only when using XCDR encoding and the Java language API for mutable types and either of the following conditions is met: • The key is not a primitive type, and the maximum serialized size of a sample for the type exceeds 65KB. • At least one key member has a member ID greater than 16128. To be compatible with the XTypes specification, this bit must be unset. This bit is unset by default. See **NDDS_CONFIG_↩ XTYPES_COMPLIANCE_MASK_DEFAULT** (p. 735)
NDDS_CONFIG_XTYPES_PYTHON_INT8_730_BIT	Set this bit to interoperate with Connext 7.3.0 Python applications that published or subscribed to types with int8 members. Value: 0x00000400 When this bit is set, the TypeCode generation for a Python type that contains an int8 member will match the one produced by the Python API in Connext 7.3.0. However, this behavior is not compliant with the XTypes specification, and Python types with int8 members won't interoperate with other APIs. When this bit is unset, the TypeCode generation for a Python type that contains an int8 member will be done according to the XTypes specification, allowing interoperability with all the APIs except Python in Connext 7.3.0. To be compatible with the XTypes specification, this bit must be unset. This bit is unset by default. See **NDDS_CONFIG_↩ XTYPES_COMPLIANCE_MASK_DEFAULT** (p. 735)

Enumerator

NDDS_CONFIG_XTYPES_MUST_UNDERSTAND_↩ IN_KEY_AND_DISCRIMINATOR_BIT	When this bit is set, the parameter ID for members that are part of the key, or for the discriminator of a union, will have the parameter ID's FLAG_MUST_UNDERSTAND bit set when serialized using XCDR2. Value: 0x00000800 To be compatible with the XTypes specification, this bit must be set. This bit only applies when you are using XCDR2. (For XCDR, the parameter ID's FLAG_MUST_UNDERSTAND bit will never be set for key members and union discriminators, because doing so would change the key hash computation and disrupt communication with older versions of RTI Connext.) This bit is unset by default. See **NDDS_CONFIG_↩ XTYPES_COMPLIANCE_MASK_DEFAULT** (p. 735)

7.216.4.2 NDDS_Config_RpcComplianceMaskBits

enum **NDDS_Config_RpcComplianceMaskBits**

The bits for the **NDDS_Config_RpcComplianceMask** (p. 737) with different aspects of RPC and Request Reply.

Enumerator

NDDS_CONFIG_RPC_USE_STANDARD_RELATED↩ _SAMPLE_PID_BIT	This bit configures a standard value for PID_RELATED_SAMPLE_IDENTITY. Value: 0x00000001 Set this bit to be compliant with the OMG DDS-RPC specification. The PID_RELATED_SAMPLE_IDENTITY will be 0x0083 as is defined as is defined in the RPC over DDS OMG specification. See https://www.omg.org/spec/DDS-RPC Using **NDDS_CONFIG_RPC_USE_STANDARD_↩ RELATED_SAMPLE_PID_BIT** (p. 745) will affect both RPC and Request-Reply communtications. The PID_RELATED_SAMPLE_IDENTITY is a parameter identifier (PID) used by the RTPS wire protocol and required for Request-Reply and RPC communications. By default, this PID takes a non-standard value that is compatible with previous versions of Connext. This bit is unset by default. See **NDDS_CONFIG_↩ RPC_COMPLIANCE_MASK_DEFAULT** (p. 736)

7.217 Heap Monitoring

Monitor memory allocations done by the middleware on the native heap.

Classes

- class **NDDSUtilityHeapMonitoring**

  Heap Monitoring APIs.

Enumerations

- enum **NDDS_Utility_HeapMonitoringSnapshotContentFormat** {
  NDDS_UTILITY_HEAP_MONITORING_SNAPSHOT_CONTENT_BIT_TOPIC ,
  NDDS_UTILITY_HEAP_MONITORING_SNAPSHOT_CONTENT_BIT_FUNCTION ,
  NDDS_UTILITY_HEAP_MONITORING_SNAPSHOT_CONTENT_BIT_ACTIVITY ,
  NDDS_UTILITY_HEAP_MONITORING_SNAPSHOT_CONTENT_DEFAULT ,
  NDDS_UTILITY_HEAP_MONITORING_SNAPSHOT_CONTENT_MINIMAL }

  Bitmap used to decide which information of the snapshot will be displayed.

7.217.1 Detailed Description

Monitor memory allocations done by the middleware on the native heap.

RTI Connext allows you to monitor the memory allocations done by the middleware on the native heap. This feature can be used to analyze and debug unexpected memory growth.

After **NDDSUtilityHeapMonitoring::enable** (p. 2114) is called, you may invoke **NDDSUtilityHeapMonitoring::take_↩ heap_snapshot** (p. 2115) to save the current heap memory usage to a file. By comparing two snapshots, you can tell if new memory has been allocated and, in many cases, where.

7.217.2 Enumeration Type Documentation

Enumerator

7.217.2.1 NDDS_Utility_HeapMonitoringSnapshotContentFormat

enum **NDDS_Utility_HeapMonitoringSnapshotContentFormat**

Bitmap used to decide which information of the snapshot will be displayed.

Enumerator

NDDS_UTILITY_HEAP_MONITORING_SNAPSHOT↩ _CONTENT_BIT_TOPIC	Add the topic to the snapshot of heap monitoring.
NDDS_UTILITY_HEAP_MONITORING_SNAPSHOT↩ _CONTENT_BIT_FUNCTION	Add the function name to the snapshot of heap monitoring.
NDDS_UTILITY_HEAP_MONITORING_SNAPSHOT↩ _CONTENT_BIT_ACTIVITY	Add the activity context to the snapshot of heap monitoring. The user can select the information that will be part of the activity context by using the API **NDDSConfigActivityContext::set_attribute_mask** (p. 733).
NDDS_UTILITY_HEAP_MONITORING_SNAPSHOT↩ _CONTENT_DEFAULT	Add all the optional attributes to the snapshot of heap monitoring.
NDDS_UTILITY_HEAP_MONITORING_SNAPSHOT↩ _CONTENT_MINIMAL	Not add any optional attribute to the snapshot of heap monitoring.

7.218 Network Capture

Save network traffic into a capture file for further analysis.

Classes

- struct **NDDS_Utility_NetworkCaptureParams_t**

  Input parameters for starting network capture.
- class **NDDSUtilityNetworkCapture**

  Network Capture APIs.

Macros

- #define **NDDS_UTILITY_NETWORK_CAPTURE_CONTENT_MASK_DEFAULT**

  *Default mask for **NDDS_Utility_NetworkCaptureContentKind** (p. 752): do not remove any content.*
- #define **NDDS_UTILITY_NETWORK_CAPTURE_CONTENT_MASK_NONE**

  The RTPS frames in the capture file will be saved as they are.
- #define **NDDS_UTILITY_NETWORK_CAPTURE_CONTENT_MASK_ALL**

The RTPS frames in the capture file will not include user data (either plain or encrypted).

- #define **NDDS_UTILITY_NETWORK_CAPTURE_TRAFFIC_MASK_DEFAULT**

  *Default mask for **NDDS_Utility_NetworkCaptureTrafficKindMask** (p. 751).*

- #define **NDDS_UTILITY_NETWORK_CAPTURE_TRAFFIC_MASK_NONE**

  Do not capture any traffic.

- #define **NDDS_UTILITY_NETWORK_CAPTURE_TRAFFIC_MASK_ALL**

  Capture all traffic (both inbound and outbound).

Typedefs

- typedef **DDS_Long NDDS_Utility_NetworkCaptureContentKindMask**

  Mask that indicates a combination of content types.

- typedef **DDS_Long NDDS_Utility_NetworkCaptureTrafficKindMask**

  Mask that indicates the traffic direction to capture.

- typedef struct **NDDS_Utility_NetworkCaptureParams_t NDDS_Utility_NetworkCaptureParams_t**

  Input parameters for starting network capture.

Enumerations

- enum **NDDS_Utility_NetworkCaptureContentKind** {
  NDDS_UTILITY_NETWORK_CAPTURE_CONTENT_USER_SERIALIZED_DATA ,
  NDDS_UTILITY_NETWORK_CAPTURE_CONTENT_ENCRYPTED_DATA }

  Bitmap used to specify a content type, i.e., a part of the RTPS frame.

- enum **NDDS_Utility_NetworkCaptureTrafficKind** {
  NDDS_UTILITY_NETWORK_CAPTURE_TRAFFIC_OUT ,
  NDDS_UTILITY_NETWORK_CAPTURE_TRAFFIC_IN }

  Bitmap used to specify whether we want to capture inbound or outbound traffic.

Variables

- const **NDDS_Utility_NetworkCaptureParams_t NDDS_UTILITY_NETWORK_CAPTURE_PARAMETERS_↩ DEFAULT**

  Default parameters used in Network Capture.

7.218.1 Detailed Description

Save network traffic into a capture file for further analysis.

RTI Connext allows you to capture the network traffic that one or more DomainParticipants send or receive. This feature can be used to analyze and debug communication problems between your DDS applications. When network capture is enabled, each DomainParticipant will generate a pcap-based file that can then be opened by a packet analyzer like Wireshark, provided the right dissectors are installed.

To some extent, network capture can be used as an alternative to existing pcap-based network capture software (such as Wireshark). This will be the case when you are only interested in analyzing the traffic a DomainParticipant

sends/receives. In this scenario, network capture will actually have some advantages over using more general pcap-based network capture applications: RTI's network capture includes additional information such as security-related data; it also removes information that is not needed, such as user data, when you want to reduce the capture size. That said, RTI's network capture is not a replacement for other pcap-based network capture applications: it only captures the traffic exchanged by the DomainParticipants, but it does not capture any other traffic exchanged through the system network interfaces.

To capture network traffic **NDDSUtilityNetworkCapture::enable** (p. 2118) must be invoked before creating any DomainParticipant. Similarly, **NDDSUtilityNetworkCapture::disable** (p. 2119) must be called after deleting all participants. In between these calls, you may start, stop, pause or resume capturing traffic for one or all participants.

7.218.2 Capturing

Shared Memory Traffic

Every RTPS frame in network capture has a source and a destination associated with it. In the case of shared memory traffic, a process identifier and a port determine the source and destination endpoints.

Access to the process identifier (PID) of the source for inbound traffic requires changes in the shared memory segments. These changes would break shared memory compatibility with previous versions of RTI Connext. For this reason, by default, network capture will not populate the value of the source PID for inbound shared memory traffic.

If interoperability with previous versions of RTI Connext is not necessary, you can generate capture files containing the source PID for inbound traffic. To do so, configure the value of the *'dds.transport.minimum_compatibility_version'* property to 6.1.0. (See **DDS_PropertyQosPolicy** (p. 1260)).

```
<domain_participant_qos>

    <property>

        <value>

            <element>

                <name>dds.transport.minimum_compatibility_version</name>

                <value>6.1.0</value>

                <propagate>false</propagate>

            </element>

        </value>

    </property>

</domain_participant_qos>
```

This property is never propagated, so it must be consistently configured throughout the whole system.

Note: Changing the value of this property affects the type of shared memory segments that RTI Connext uses. For that reason, you may see the following warning, resulting from leftover shared memory segments:

```
[0xC733A001,0xB248F671,0xAEC4A0C1:0x000001C1{Domain=200}|CREATE DP|ENABLE]
       NDDS_Transport_Shmem_is_segment_compatible:incompatible shared memory protocol detected.

Current version 4.0 not compatible with 2.0.
```

The leftover shared memory segments can be removed using the `ipcrm` command. See `https://community.rti.com/kb/what-are-possible-solutions-common-shared-memory-issues` for more information.

7.218.3 Macro Definition Documentation

7.218.3.1 NDDS_UTILITY_NETWORK_CAPTURE_CONTENT_MASK_DEFAULT

```
#define NDDS_UTILITY_NETWORK_CAPTURE_CONTENT_MASK_DEFAULT
```

Default mask for **NDDS_Utility_NetworkCaptureContentKind** (p. 752): do not remove any content.

It is equivalent to **NDDS_UTILITY_NETWORK_CAPTURE_CONTENT_MASK_NONE** (p. 750).

[default] Do not remove any content.

7.218.3.2 NDDS_UTILITY_NETWORK_CAPTURE_CONTENT_MASK_NONE

```
#define NDDS_UTILITY_NETWORK_CAPTURE_CONTENT_MASK_NONE
```

The RTPS frames in the capture file will be saved as they are.

7.218.3.3 NDDS_UTILITY_NETWORK_CAPTURE_CONTENT_MASK_ALL

```
#define NDDS_UTILITY_NETWORK_CAPTURE_CONTENT_MASK_ALL
```

The RTPS frames in the capture file will not include user data (either plain or encrypted).

Its value is the result of setting the bits for removing user data and removing encrypted data: (**NDDS_↩ UTILITY_NETWORK_CAPTURE_CONTENT_USER_SERIALIZED_DATA** (p. 752)) | **NDDS_UTILITY_NETWORK_↩ CAPTURE_CONTENT_ENCRYPTED_DATA** (p. 752))

7.218.3.4 NDDS_UTILITY_NETWORK_CAPTURE_TRAFFIC_MASK_DEFAULT

```
#define NDDS_UTILITY_NETWORK_CAPTURE_TRAFFIC_MASK_DEFAULT
```

Default mask for **NDDS_Utility_NetworkCaptureTrafficKindMask** (p. 751).

It is equivalent to **NDDS_UTILITY_NETWORK_CAPTURE_TRAFFIC_MASK_ALL** (p. 751).

[default] Capture all traffic: inbound and outbound.

7.218.3.5 NDDS_UTILITY_NETWORK_CAPTURE_TRAFFIC_MASK_NONE

```
#define NDDS_UTILITY_NETWORK_CAPTURE_TRAFFIC_MASK_NONE
```

Do not capture any traffic.

7.218.3.6 NDDS_UTILITY_NETWORK_CAPTURE_TRAFFIC_MASK_ALL

```
#define NDDS_UTILITY_NETWORK_CAPTURE_TRAFFIC_MASK_ALL
```

Capture all traffic (both inbound and outbound).

The value is equal to setting both the input and output bits of the mask: (**NDDS_UTILITY_NETWORK_CAPTURE_↩ TRAFFIC_OUT** (p. 752) | **NDDS_UTILITY_NETWORK_CAPTURE_TRAFFIC_OUT** (p. 752)).

7.218.4 Typedef Documentation

7.218.4.1 NDDS_Utility_NetworkCaptureContentKindMask

```
typedef DDS_Long NDDS_Utility_NetworkCaptureContentKindMask
```

Mask that indicates a combination of content types.

The masks are based on a combination (or only one) of the **NDDS_Utility_NetworkCaptureContentKind** (p. 752) bitmaps.

See also

NDDS_Utility_NetworkCaptureContentKind (p. 752)

7.218.4.2 NDDS_Utility_NetworkCaptureTrafficKindMask

```
typedef DDS_Long NDDS_Utility_NetworkCaptureTrafficKindMask
```

Mask that indicates the traffic direction to capture.

The masks are based on a combination (or only one) of the **NDDS_Utility_NetworkCaptureTrafficKind** (p. 752) bitmaps.

See also

NDDS_Utility_NetworkCaptureTrafficKind (p. 752)

7.218.4.3 NDDS_Utility_NetworkCaptureParams_t

```
typedef struct NDDS_Utility_NetworkCaptureParams_t NDDS_Utility_NetworkCaptureParams_t
```

Input parameters for starting network capture.

7.218.5 Enumeration Type Documentation

7.218.5.1 NDDS_Utility_NetworkCaptureContentKind

```
enum NDDS_Utility_NetworkCaptureContentKind
```

Bitmap used to specify a content type, i.e., a part of the RTPS frame.

Several values can be combined. Read **NDDS_Utility_NetworkCaptureContentKindMask** (p. 751) for typical combinations.

Enumerator

NDDS_UTILITY_NETWORK_CAPTURE_CONTENT_USER_SERIALIZED_DATA	The serialized data coming from a user.
NDDS_UTILITY_NETWORK_CAPTURE_CONTENT_ENCRYPTED_DATA	The encrypted user data.

7.218.5.2 NDDS_Utility_NetworkCaptureTrafficKind

```
enum NDDS_Utility_NetworkCaptureTrafficKind
```

Bitmap used to specify whether we want to capture inbound or outbound traffic.

Several values can be combined. Read **NDDS_Utility_NetworkCaptureTrafficKindMask** (p. 751) for typical combinations.

Enumerator

NDDS_UTILITY_NETWORK_CAPTURE_TRAFFIC_OUT	Do not capture outbound traffic.
NDDS_UTILITY_NETWORK_CAPTURE_TRAFFIC_IN	Do not capture inbound traffic.

7.218.6 Variable Documentation

7.218.6.1 NDDS_UTILITY_NETWORK_CAPTURE_PARAMETERS_DEFAULT

```
const NDDS_Utility_NetworkCaptureParams_t NDDS_UTILITY_NETWORK_CAPTURE_PARAMETERS_DEFAULT [extern]
```

Default parameters used in Network Capture.

The default parameters can be used as initilizers to ensure that new objects do not have uninitialized contents.

Initialization of a **NDDS_Utility_NetworkCaptureParams_t** (p. 2087) type can be done either by assigning to the default parameters:

```
struct NDDS_Utility_NetworkCaptureParams_t params = NDDS_UTILITY_NETWORK_CAPTURE_PARAMETERS_DEFAULT;
```

Or equivalently, by using its initializer method:

```
struct NDDS_Utility_NetworkCaptureParams_t params;

NDDS_Utility_NetworkCaptureParams_t_initialize(&params);
```

In either case, do not forget to finalize the parameters when you don't need them any more:

```
NDDS_Utility_NetworkCaptureParams_t_finalize(&params);
```

The values that are used by default for the parameters can be found in the description of each of the members in **NDDS_Utility_NetworkCaptureParams_t** (p. 2087).

7.219 Other Utilities

Other Utilities, such as **NDDSUtility::spin** (p. 2111).

Classes

- class **NDDSLM**

  Other Utilities APIs.
- class **NDDSUtility**

  Other Utilities APIs.

7.219.1 Detailed Description

Other Utilities, such as **NDDSUtility::spin** (p. 2111).

7.220 Octet Buffer Support

<<***extension***>> *(p. 432)* Octet buffer creation, cloning, and deletion.

Functions

- unsigned char * **DDS_OctetBuffer_alloc** (unsigned int size)

  Create a new empty OctetBuffer that can hold up to `size` octets.
- unsigned char * **DDS_OctetBuffer_dup** (const unsigned char *buffer, unsigned int size)

  Clone an OctetBuffer.
- void **DDS_OctetBuffer_free** (unsigned char *buffer)

  Delete an OctetBuffer.

7.220.1 Detailed Description

<<***extension***>> *(p. 432)* Octet buffer creation, cloning, and deletion.

The methods in this class ensure consistent cross-platform implementations for OctetBuffer creation (**DDS_Octet↩ Buffer_alloc()** (p. 755)), deletion (**DDS_OctetBuffer_free()** (p. 756)), and cloning (**DDS_OctetBuffer_dup()** (p. 755)) that preserve the mutable value type semantics. These are to be viewed as methods that define an `OctetBuffer` class whose data is represented by a '`unsigned` char*'.

7.220.2 Conventions

The following conventions govern the memory management of OctetBuffers in RTI Connext.

- The DDS implementation ensures that when value types containing OctetBuffers are passed back and forth to the DDS APIs, the OctetBuffers are created/deleted/cloned using the `OctetBuffer` class methods.
  - Value types containing OctetBuffers have ownership of the contained OctetBuffer. Thus, when a value type is deleted, the contained octet buffer field is also deleted.
- The user must ensure that when value types containing OctetBuffers are passed back and forth to the DDS APIs, the OctetBuffers are created/deleted/cloned using the OctetBuffer class methods.

The representation of an OctetBuffer in C/C++ unfortunately does not allow programs to detect how much memory has been allocated for a OctetBuffer. RTI Connext must therefore make some assumptions when a user requests that a OctetBuffer be copied into. The following rules apply when RTI Connext is copying into an OctetBuffer.

- If the 'unsigned char*' is NULL, RTI Connext will allocate a new OctetBuffer on behalf of the user. *To avoid leaking memory, you must ensure that the OctetBuffer will be freed (see **Usage** (p. 755) below) in C. For C++, the destructor of the valuetype containing the OctetBuffer will free it automatically.*.
- If the 'unsigned char*' is not NULL, RTI Connext will assume that you are managing the OctetBuffer's memory yourself and have allocated enough memory to store the OctetBuffer to be copied. *RTI Connext will copy into your memory; to avoid memory corruption, be sure to allocate enough of it.* Also, *do not pass structures containing junk pointers into RTI Connext*; you are likely to crash.

7.220.3 Usage

This requirement can generally be assured by adhering to the following *idiom* for manipulating OctetBuffers.

```
Always use
     DDS_OctetBuffer_alloc() to create,
     DDS_OctetBuffer_dup() to clone,
     DDS_OctetBuffer_free() to delete
 a 'unsigned char*' that is passed back and forth between
 user code and the DDS C/C++ APIs.
```

Not adhering to this idiom can result in bad pointers, and incorrect memory being freed.

In addition, the user code should be vigilant to avoid memory leaks. It is good practice to:

- Balance occurrences of **DDS_OctetBuffer_alloc()** (p. 755), with matching occurrences of **DDS_OctetBuffer_↩ free()** (p. 756) in the code.
- Finalize value types containing OctetBuffer. In C++ the destructor accomplishes this automatically. in C, explicit "destructor" functions are provided.

7.220.4 Function Documentation

7.220.4.1 DDS_OctetBuffer_alloc()

```
unsigned char * DDS_OctetBuffer_alloc (
             unsigned int size )
```

Create a new empty OctetBuffer that can hold up to `size` octets.

An OctetBuffer created by this method must be deleted using **DDS_OctetBuffer_free()** (p. 756).

This function will allocate enough memory to hold an OctetBuffer of `size` octets.

Parameters

size	<<***in***>> *(p. 433)* Size of the buffer.

Returns

A newly created non-NULL OctetBuffer upon success or NULL upon failure.

Referenced by **DDS_KeyedOctets::DDS_KeyedOctets()**, and **DDS_Octets::DDS_Octets()**.

7.220.4.2 DDS_OctetBuffer_dup()

```
unsigned char * DDS_OctetBuffer_dup (
            const unsigned char * buffer,
            unsigned int size )
```

Clone an OctetBuffer.

```
An OctetBuffer created by this method must be deleted using
DDS_OctetBuffer_free()
```

Parameters

buffer	<<***in***>> *(p. 433)* The OctetBuffer to duplicate.
size	<<***in***>> *(p. 433)* Size of the OctetBuffer to duplicate.

Returns

If `src` == NULL or size <0, this method always returns NULL. Otherwise, upon success it returns a newly created OctetBuffer whose value is `src`; upon failure it returns NULL.

7.220.4.3 DDS_OctetBuffer_free()

```
void DDS_OctetBuffer_free (
            unsigned char * buffer )
```

Delete an OctetBuffer.

Precondition

`buffer` must be either NULL, or must have been created using **DDS_OctetBuffer_alloc()** (p. 755), **DDS_↩ OctetBuffer_dup()** (p. 755)

Parameters

buffer	<<***in***>> *(p. 433)* The buffer to delete.

Referenced by **DDS_KeyedOctets::∼DDS_KeyedOctets()**, and **DDS_Octets::∼DDS_Octets()**.

7.221 SampleProcessor

<<***experimental***>> *(p. 432)* <<***extension***>> *(p. 432)* Utility to concurrently read and process the data samples received by **DDSDataReader** (p. 1556).

*<<**experimental**>> (p. 432) <<**extension**>> (p. 432)* Utility to concurrently read and process the data samples received by **DDSDataReader** (p. 1556).

7.222 Sequence Support

The **FooSeq** (p. 1965) interface allows you to work with variable-length collections of homogeneous data.

Modules

- **Built-in Sequences**

  Defines sequences of primitive data type. .

Classes

- struct **FooSeq**

  *<<**interface**>> (p. 433) <<**generic**>> (p. 433) A type-safe, ordered collection of elements. The type of these elements is referred to in this documentation as **`Foo`** (p. 1916).*

7.222.1 Detailed Description

The **FooSeq** (p. 1965) interface allows you to work with variable-length collections of homogeneous data.

This interface is instantiated for each concrete element type in order to provide compile-time type safety to applications. The **Built-in Sequences** (p. 169) are pre-defined instantiations for the primitive data types.

When you use the rtiddsgen code generation tool, it will automatically generate concrete sequence instantiations for each of your own custom types.

7.223 String Support

*<<**extension**>> (p. 432)* String creation, cloning, assignment, and deletion.

Functions

- char * **DDS_String_alloc** (size_t length)

  Create a new empty string that can hold up to `length` *characters.*
- char * **DDS_String_dup** (const char *str)

  Clone a string. Creates a new string that duplicates the value of `string`*.*
- char * **DDS_String_replace** (char **string_ptr, const char *new_value)

  Assign a new value to a string. Replaces the string pointed to by `string_ptr`*, with a string whose value is* `new_value`*.*
- void **DDS_String_free** (char *str)

  Delete a string.
- **DDS_Wchar** * **DDS_Wstring_alloc** (**DDS_UnsignedLong** length)

  Create a new empty string that can hold up to `length` *wide characters.*
- **DDS_UnsignedLong** **DDS_Wstring_length** (const **DDS_Wchar** *str)

  Get the number of wide characters in the given string.
- **DDS_Wchar** * **DDS_Wstring_copy** (**DDS_Wchar** *dst, const **DDS_Wchar** *src)

  Copy the source string over the destination string reallocating the space if it's necessary.
- **DDS_Wchar** * **DDS_Wstring_copy_and_widen** (**DDS_Wchar** *dst, const char *src)

  Copy the source string over the destination string, widening each character.
- **DDS_Wchar** * **DDS_Wstring_dup** (const **DDS_Wchar** *str)

  Clone a string of wide characters. Creates a new string that duplicates the value of `string`*.*
- **DDS_Wchar** * **DDS_Wstring_dup_and_widen** (const char *str)

  Clone a string of characters as a string of wide characters.
- void **DDS_Wstring_free** (**DDS_Wchar** *str)

  Delete a string.

7.223.1 Detailed Description

<<***extension***>> *(*p. *432)* String creation, cloning, assignment, and deletion.

The methods in this class ensure consistent cross-platform implementations for string creation (**DDS_String_alloc()** (p. 759)), deletion (**DDS_String_free()** (p. 762)), and cloning (**DDS_String_dup()** (p. 761)) that preserve the mutable value type semantics. These are to be viewed as methods that define a `string` class whose data is represented by a `'char*'`.

7.223.2 String Conventions

The following conventions govern the memory management of strings in RTI Connext.

- The DDS implementation ensures that when value types containing strings are passed back and forth to the DDS APIs, the strings are created/deleted/assigned/cloned using the `string` class methods.
  - Value types containing strings have ownership of the contained string. Thus, when a value type is deleted, the contained string field is also deleted.
  - **DDS_StringSeq** (p. 1364) is a value type that contains strings; it owns the memory for the contained strings. When a **DDS_StringSeq** (p. 1364) is assigned or deleted, the contained strings are also assigned or deleted respectively.

- The user must ensure that when value types containing strings are passed back and forth to the DDS APIs, the strings are created/deleted/assigned/cloned using the String class methods.

The representation of a string in C/C++ unfortunately does not allow programs to detect how much memory has been allocated for a string. RTI Connext must therefore make some assumptions when a user requests that a string be copied into. The following rules apply when RTI Connext is copying into a string or string sequence:

- If the 'char∗' is NULL, RTI Connext will log a warning and allocate a new string on behalf of the user. *To avoid leaking memory, you must ensure that the string will be freed (see **Usage** (p. 759) below).*
- If the 'char∗' is not NULL, RTI Connext will assume that you are managing the string's memory yourself and have allocated enough memory to store the string to be copied. *RTI Connext will copy into your memory; to avoid memory corruption, be sure to allocate enough of it.* Also, *do not pass structures containing junk pointers into RTI Connext*; you are likely to crash.

7.223.3 Usage

This requirement can generally be assured by adhering to the following *idiom* for manipulating strings.

```
Always use
     DDS_String_alloc() to create,
     DDS_String_dup() to clone,
     DDS_String_free() to delete
 a string 'char*' that is passed back and forth between
 user code and the DDS C/C++ APIs.
```

Not adhering to this idiom can result in bad pointers, and incorrect memory being freed.

In addition, the user code should be vigilant to avoid memory leaks. It is good practice to:

- Balance occurrences of **DDS_String_alloc()** (p. 759), **DDS_String_dup()** (p. 761), with matching occurrences of **DDS_String_free()** (p. 762) in the code.
- Finalize value types containing strings. In C++ the destructor accomplishes this automatically. in C, explicit "destructor" functions are provided; these functions are typically called "finalize."

Note

When dealing with the **DDS_PublishModeQosPolicy::flow_controller_name** (p. 1281) and **DDS_Multi↩ ChannelQosPolicy::filter_name** (p. 1215) fields, we advise taking a look at the sample code for how to properly assign new values and free the old ones.

See also

DDS_StringSeq (p. 1364)

7.223.4 Function Documentation

7.223.4.1 DDS_String_alloc()

```
char * DDS_String_alloc (
            size_t length )
```

Create a new empty string that can hold up to `length` characters.

A string created by this method must be deleted using **DDS_String_free()** (p. 762).

This function will allocate enough memory to hold a string of `length` characters, **plus** one additional byte to hold the NULL terminating character.

Parameters

length	<<***in***>> *(*p. *433)* Capacity of the string.

Returns

A newly created non-NULL string upon success or NULL upon failure.

Examples

HelloWorld.cxx.

Referenced by **DDS_KeyedOctets::DDS_KeyedOctets()**, and **DDS_KeyedString::DDS_KeyedString()**.

7.223.4.2 DDS_String_dup()

```
char * DDS_String_dup (
            const char * str )
```

Clone a string. Creates a new string that duplicates the value of `string`.

A string created by this method must be deleted using **DDS_String_free()** (p. 762)

Parameters

str	<<***in***>> *(*p. *433)* The string to duplicate.

Returns

If `string` == NULL, this method always returns NULL. Otherwise, upon success it returns a newly created string whose value is `string`; upon failure it returns NULL.

7.223.4.3 DDS_String_replace()

```
char * DDS_String_replace (
            char ** string_ptr,
            const char * new_value )
```

Assign a new value to a string. Replaces the string pointed to by `string_ptr`, with a string whose value is `new_value`.

A string created by this method must be deleted using **DDS_String_free()** (p. 762).

This function is most commonly used when manipulating string sequences, **DDS_StringSeq** (p. 1364).

Precondition

`string_ptr` be a non-NULL pointer. ∗string_ptr must be either NULL, or a string created using **DDS_String_alloc()** (p. 759) or **DDS_String_dup()** (p. 761), or **DDS_String_replace()** (p. 761).

Parameters

string_ptr	<<***inout***>> *(p. 433)* Pointer to the string to replace.
new_value	<<***in***>> *(p. 433)* The value of the replacement string.

Returns

If `new_value` = NULL, this method always returns NULL. Otherwise, upon success it returns ∗string_ptr whose value is `new_value`; upon failure it returns NULL.

Postcondition

If `new_value` = NULL, ∗string_ptr == NULL. Otherwise, upon success `string_ptr` contains a pointer to a string whose value is `new_value`; upon failure, `string_ptr` is left unchanged.

7.223.4.4 DDS_String_free()

```
void DDS_String_free (
             char * str )
```

Delete a string.

Precondition

`string` must be either NULL, or must have been created using **DDS_String_alloc()** (p. 759), **DDS_String_dup()** (p. 761)

Parameters

str	<<***in***>> *(p. 433)* The string to delete.

Examples

HelloWorld.cxx.

Referenced by **DDS_KeyedOctets::DDS_KeyedOctets()**, **DDS_KeyedString::DDS_KeyedString()**, **DDS_KeyedOctets::∼DDS_KeyedOctets()**, and **DDS_KeyedString::∼DDS_KeyedString()**.

7.223.4.5 DDS_Wstring_alloc()

```
DDS_Wchar * DDS_Wstring_alloc (
            DDS_UnsignedLong length )
```

Create a new empty string that can hold up to `length` wide characters.

A string created by this method must be deleted using **DDS_Wstring_free()** (p. 765)

This function will allocate enough memory to hold a string of `length` characters, **plus** one additional wide character to hold the NULL terminator.

Parameters

length	<<***in***>> *(p. 433)* Capacity of the string.

Returns

A newly created non-NULL string upon success or NULL upon failure.

7.223.4.6 DDS_Wstring_length()

```
DDS_UnsignedLong DDS_Wstring_length (
            const  DDS_Wchar * str )
```

Get the number of wide characters in the given string.

The result does not count the terminating zero character.

Parameters

str	<<***in***>> *(p. 433)* A non-NULL string.

Returns

The number of wide characters in the string.

7.223.4.7 DDS_Wstring_copy()

```
DDS_Wchar * DDS_Wstring_copy (
            DDS_Wchar * dst,
           const  DDS_Wchar * src )
```

Copy the source string over the destination string reallocating the space if it's necessary.

Parameters

dst	
src	

Returns

dst

7.223.4.8 DDS_Wstring_copy_and_widen()

```
DDS_Wchar * DDS_Wstring_copy_and_widen (
            DDS_Wchar * dst,
            const char * src )
```

Copy the source string over the destination string, widening each character.

Parameters

dst	<<***in***>> *(*p. *433)* A non-NULL string to be overwritten by src.
src	<<***in***>> *(*p. *433)* A non-NULL string to be copied over dst

Returns

dst

7.223.4.9 DDS_Wstring_dup()

```
DDS_Wchar * DDS_Wstring_dup (
            const  DDS_Wchar * str )
```

Clone a string of wide characters. Creates a new string that duplicates the value of string.

A string created by this method must be deleted using **DDS_Wstring_free()** (p. 765).

Parameters

str	<<***in***>> *(*p. *433)* The string to duplicate.

Returns

If `string` == NULL, this method always returns NULL. Otherwise, upon success it returns a newly created string whose value is `string`; upon failure it returns NULL.

7.223.4.10 DDS_Wstring_dup_and_widen()

```
DDS_Wchar * DDS_Wstring_dup_and_widen (
            const char * str )
```

Clone a string of characters as a string of wide characters.

A string created by this method must be deleted using **DDS_Wstring_free()** (p. 765)

Parameters

str	*<<**in**>> (p. 433)* The string to duplicate.

Returns

If `string` == NULL, this method always returns NULL. Otherwise, upon success it returns a newly created string whose value is `string`; upon failure it returns NULL.

7.223.4.11 DDS_Wstring_free()

```
void DDS_Wstring_free (
             DDS_Wchar * str )
```

Delete a string.

Precondition

`string` must either NULL, or must have been created using **DDS_Wstring_alloc()** (p. 762), **DDS_Wstring_dup()** (p. 764), or DDS_Wstring_replace()

Parameters

str	*<<**in**>> (p. 433)* The string to delete.

7.224 FlatData Builders

A Builder allows creating and initializing variable-size data.

Classes

- class **rti::flat::AggregationBuilder**
  Base class of struct and union builders.
- class **rti::flat::UnionBuilder< Discriminator >**
  Base class of builders for user-defined mutable unions.
- class **MyFlatMutableBuilder**
  Represents the Builder for an arbitrary user-defined mutable type.
- class **MyFlatUnionBuilder**
  Represents the Builder for an arbitrary user-defined mutable IDL union.
- class **rti::flat::AbstractBuilder**
  *Base class of all **Builders** (p. 766).*
- class **rti::flat::AbstractListBuilder< DHeaderGen >**
  Base class of all array and sequence builders.
- class **rti::flat::MutableArrayBuilder< ElementBuilder, N >**
  Builds an array member of variable-size elements.
- class **rti::flat::AbstractSequenceBuilder< DHeaderGen >**
  Base class of Builders for sequence members.
- class **rti::flat::MutableSequenceBuilder< ElementBuilder >**
  Builds a sequence member of variable-size elements.
- class **rti::flat::FinalSequenceBuilder< ElementOffset >**
  Builds a sequence member of fixed-size elements.
- class **rti::flat::PrimitiveSequenceBuilder< T >**
  Builds a sequence of primitive members.
- class **rti::flat::StringBuilder**
  Builds a string.

Functions

- template<typename TopicType >
  rti::flat::flat_type_traits< TopicType >::builder **rti::flat::build_data** (typename TopicType::DataWriter *writer)
  Begins building a new sample.
- template<typename BuilderType >
  void **rti::flat::discard_builder** (typename **rti::flat::flat_type_traits**< BuilderType >::flat_type::DataWriter *writer, BuilderType &builder)
  Discards a sample builder.

7.224.1 Detailed Description

A Builder allows creating and initializing variable-size data.

Builders allow creating variable-size **FlatData samples** (p. 769), as described in **Publishing FlatData** (p. 777).

There are the following Builder types:

Category	Builder type
User types	For example: • **MyFlatMutableBuilder** (p. 2018), a mutable struct • **MyFlatUnionBuilder** (p. 2032), a union • Final structs are fixed-size and do not have a Builder (see for example **MyFlatMutable↩ Builder::add_my_final()** (p. 2023)).
Arrays	• **rti::flat::MutableArrayBuilder** (p. 2008) • Arrays of final elements are fixed-size and do not have a Builder (see for example **MyFlat↩ MutableBuilder::add_my_final_array()** (p. 2023)).
Sequences	• **rti::flat::MutableSequenceBuilder** (p. 2012) for sequences of mutable elements • **rti::flat::FinalSequenceBuilder** (p. 1913) for sequences of final elements • **rti::flat::PrimitiveSequenceBuilder** (p. 2135) for sequences of primitive elements • **rti::flat::StringBuilder** (p. 2214)
Primitive types	• Primitive members are added using one of the member functions of the Builder of the type that contains them. See for example **MyFlatMutableBuilder::add_my_primitive()** (p. 2022)<

Builder for user types, such as **MyFlatMutableBuilder** (p. 2018), can behave as **sample builders**, when they build a sample, or **member builders** when they build a member for another Builder (such as the Builder returned by **MyFlat↩ MutableBuilder::build_my_mutable()** (p. 2024)). All the other Builder types (sequences, arrays) are always member builders (such as the Builder returned by **MyFlatMutableBuilder::build_my_final_seq()** (p. 2023)).

Builders are **move-only** types. They cannot be copied, only moved. When you assign a Builder as the return value of a function, this happens automatically.

```
// the return value of 'build_data' is moved into 'builder'
MyFlatMutableBuilder builder = rti::flat::build_data(writer);
```

To explicitly move a Builder variable in the Traditional C++ API, call its **move()** (p. 195) member function

7.224.2 Builder Error Management

The Traditional C++ API reports Builder errors by setting a flag that can be checked with the member function **check↩ _failure()** (p. 789). After each Builder operation, `check_failure()` must be called to check (and clear) the error flag.

In addition, in case of error, operations that return another Builder, such as **MyFlatMutableBuilder::build_my_↩ mutable()** (p. 2024), return an invalid Builder (**is_valid()** (p. 788) is `false`). Operations that return an Offset, such as **MyFlatMutableBuilder::add_my_final()** (p. 2023) return a "null" Offset when they fail (see **Offset Error Management** (p. 773)).

See also

MyFlatMutableBuilder (p. 2018) for more information specific to builders for user types.

7.224.3 Function Documentation

7.224.3.1 build_data()

```
template<typename TopicType >
rti::flat::flat_type_traits< TopicType >::builder rti::flat::build_data (
            typename TopicType::DataWriter * writer )
```

Begins building a new sample.

Template Parameters

TopicType	A FlatData mutable type that corresponds to the type of the DataWriter argument.

Parameters

writer	The writer that will be used to write this sample.

Returns

The Builder to build the sample. For example if TopicType is **MyFlatMutable** (p. 770), this function returns a **MyFlatMutableBuilder** (p. 2018). In case of error, this function returns an **invalid Builder** (p. 788)

Once you have completed the sample, call **finish_sample()** (p. 2021) to obtain a **MyFlatMutable** (p. 770) sample that can be written with `writer`.

If the building of this sample needs to be aborted before calling finish_sample(), use **rti::flat::discard_builder()** (p. 768). If, after obtaining a sample with finish_sample(), this sample is not written, then discard it with **FooDataWriter::discard_loan()** (p. 1964).

See also

FooDataWriter::get_loan (p. 1963), the function that **build_data()** (p. 768) uses to obtain the memory required to build the sample.

Publishing FlatData (p. 777)

References **DDS_BOOLEAN_TRUE**, and **DDS_RETCODE_OK**.

7.224.3.2 discard_builder()

```
template<typename BuilderType >
void rti::flat::discard_builder (
            typename  rti::flat::flat_type_traits< BuilderType >::flat_type::DataWriter * writer,
            BuilderType & builder )
```

Discards a sample builder.

Template Parameters

TopicType	An IDL-defined FlatData type that corresponds to the type of the DataWriter argument.

This function invalidates and discards a builder before it is finished() and before the sample it would have created was written.

Parameters

writer	The writer that was used to create the builder
builder	The builder, created with `rti::flat::build_data(writer);`

7.225 FlatData Samples

A Sample represents an instance of the IDL topic-type and contains the data in serialized format.

Classes

- class **rti::flat::Sample**< **OffsetType** >

  The generic definition of FlatData topic-types.
- struct **rti::flat::flat_type_traits**< **T** >

  *Given a **Sample** (p. 2187), an Offset or a Builder, it allows obtaining the other types.*

Typedefs

- typedef **rti::flat::Sample**< **MyFlatFinalOffset** > **MyFlatFinal**

  Represents an arbitrary user-defined FlatData final IDL struct.
- typedef **rti::flat::Sample**< **MyFlatMutableOffset** > **MyFlatMutable**

  Represents an arbitrary user-defined flat mutable IDL struct.
- typedef **rti::flat::Sample**< **MyFlatUnionOffset** > **MyFlatUnion**

  Represents an arbitrary user-defined flat mutable IDL union.

7.225.1 Detailed Description

A Sample represents an instance of the IDL topic-type and contains the data in serialized format.

All FlatData topic-types are instantiations of **rti::flat::Sample** (p. 2187). This documentation uses the following three example types to illustrate the different ways FlatData IDL types map to C++:

- **MyFlatFinal** (p. 770), the type generated for a final IDL struct
- **MyFlatMutable** (p. 770), the type generated for a mutable IDL struct
- **MyFlatUnion** (p. 770), the type generated for an IDL union

7.225.2 Typedef Documentation

7.225.2.1 MyFlatFinal

typedef **rti::flat::Sample**< **MyFlatFinalOffset**> **MyFlatFinal**

Represents an arbitrary user-defined FlatData final IDL struct.

This documentation uses the following example IDL definition of MyFlatFinal:

```
@language_binding(FLAT_DATA)
@final
struct MyFlatFinal {
    long my_primitive;
    FlatFinalBar my_complex; // Another arbitrary final FlatData type
    long my_primitive_array[10];
    FlatFinalBar my_complex_array[10];
};
```

For this type, **rtiddsgen** generates:

- MyFlatFinal (this instantiation of Sample),
- **MyFlatFinalOffset** (p. 2014).

Note that, as a final FlatData type, MyFlatFinal can only contain fixed-size types, such as primitives, other final FlatData structs, and arrays of fixed-size types.

Samples of MyFlatFinal are created with **FooDataWriter::get_loan** (p. 1963). After creating a final Sample, modify its values using the **MyFlatFinalOffset** (p. 2014) returned by **root()** (p. 2189).

7.225.2.2 MyFlatMutable

typedef **rti::flat::Sample**< **MyFlatMutableOffset**> **MyFlatMutable**

Represents an arbitrary user-defined flat mutable IDL struct.

This documentation uses the following example IDL definition of MyFlatMutable:

```
@language_binding(FLAT_DATA)
@mutable
struct MyFlatMutable {
    long my_primitive;
    @optional long my_optional_primitive;
    long my_primitive_array[10];
    sequence<long, 10> my_primitive_seq;
    MyFlatFinal my_final;
    MyFlatFinal my_final_array[10];
    sequence<MyFlatFinal, 10> my_final_seq;
    FlatMutableBar my_mutable;
    FlatMutableBar my_mutable_array[10];
    sequence<FlatMutableBar, 10> my_mutable_seq;
    string<255> my_string;
    sequence<string<255>, 10> my_string_seq;
};
```

For this type, **rtiddsgen** generates:

- MyFlatMutable (this instantiation of Sample),
- **MyFlatMutableOffset** (p. 2025),
- **MyFlatMutableBuilder** (p. 2018).

As a mutable FlatData type, MyFlatMutable is not restricted to fixed-size members. Samples of MyFlatMutable are created with **rti::flat::build_data()** (p. 768), which returns a **MyFlatMutableBuilder** (p. 2018). Once built, a mutable Sample can't change in size, but the value of members that already exist can be changed using the **MyFlatMutable↩ Offset** (p. 2025) returned by **root()** (p. 2189).

7.225.2.3 MyFlatUnion

```
typedef rti::flat::Sample< MyFlatUnionOffset> MyFlatUnion
```

Represents an arbitrary user-defined flat mutable IDL union.

This documentation uses the following example IDL definition of MyFlatUnion:

```
@language_binding(FLAT_DATA)
@mutable
union MyFlatUnion switch (long) {
    case 0:
        long my_primitive;
    case 1:
    case 2:
        MyFlatMutable my_mutable;
    case 3:
        MyFlatFinal my_final;
};
```

Note

FlatData unions can only be **mutable** since unions are, by definition, variable-size types.

For this type, **rtiddsgen** generates:

- MyFlatUnion (this instantiation of Sample),
- **MyFlatUnionOffset** (p. 2035),
- **MyFlatUnionBuilder** (p. 2032).

7.226 FlatData Offsets

Offsets provide access to the members of a FlatData Sample.

Classes

- class **MyFlatFinalOffset**

  Represents the Offset to an arbitrary user-defined FlatData final IDL struct.
- class **MyFlatMutableOffset**

  Represents the Offset to an arbitrary user-defined FlatData mutable IDL struct.
- class **MyFlatUnionOffset**

  Represents the Offset to an arbitrary user-defined FlatData mutable IDL union.
- class **rti::flat::OffsetBase**

  Base class of all Offset types.
- class **rti::flat::FinalOffset**< **T** >

  The base class of all Offsets to a final struct type.
- class **rti::flat::MutableOffset**

  The base class of all Offsets to a final struct type.
- struct **rti::flat::PrimitiveConstOffset**< **T** >

  A const Offset to an optional primitive member.

- struct **rti::flat::PrimitiveOffset< T >**

  An Offset to an optional primitive member.
- class **rti::flat::SequenceIterator< E, OffsetKind >**

  Iterator for collections of Offsets.
- class **rti::flat::AbstractPrimitiveList< T >**

  Base class for Offsets to sequences and arrays of primitive types.
- class **rti::flat::PrimitiveSequenceOffset< T >**

  Offset to a sequence of primitive elements.
- class **rti::flat::PrimitiveArrayOffset< T, N >**

  Offset to an array of primitive elements.
- class **rti::flat::StringOffset**

  Offset to a string.
- class **rti::flat::AbstractAlignedList< ElementOffset >**

  Base class of Offsets to sequences and arrays of non-primitive members.
- class **rti::flat::SequenceOffset< ElementOffset >**

  Offset to a sequence of non-primitive elements.
- class **rti::flat::MutableArrayOffset< ElementOffset, N >**

  Offset to an array of variable-size elements.
- class **rti::flat::FinalArrayOffset< ElementOffset, N >**

  Offset to an array of final elements.
- class **rti::flat::FinalAlignedArrayOffset< ElementOffset, N >**

  Offset to an array of final elements.

Functions

- template<typename OffsetType >
  flat_type_traits< OffsetType >::plain_type * **rti::flat::plain_cast** (OffsetType &offset)

  Casts into an equivalent plain C++ type.
- template<typename OffsetType >
  const **flat_type_traits**< OffsetType >::plain_type * **rti::flat::plain_cast** (const OffsetType &offset)

  Const version of plain_cast.

7.226.1 Detailed Description

Offsets provide access to the members of a FlatData Sample.

An Offset represents a position within a **FlatData Sample** (p. 769) that allows accessing a member of that Sample, or the Sample's **root** (p. 2189).

Offsets can be described as **iterators**. They represent a position in a buffer, not the value itself. As such, they're lightweight objects that can be copied to point to the same data.

There are the following Offset types to access the different IDL types:

Category	Offset type
User types	For example: • **MyFlatFinalOffset** (p. 2014), an Offset to a final struct • **MyFlatMutableOffset** (p. 2025), an Offset to a mutable struct • **MyFlatUnionOffset** (p. 2035), an Offset to a union
Arrays	• **rti::flat::PrimitiveArrayOffset** (p. 2132) • **rti::flat::FinalArrayOffset** (p. 1911) (array of final elements in a final type) • **rti::flat::FinalAlignedArrayOffset** (p. 1909) (array of final elements in a mutable type) • **rti::flat::MutableArrayOffset** (p. 2010)
Sequences	• **rti::flat::PrimitiveSequenceOffset** (p. 2138) • **rti::flat::SequenceOffset** (p. 2204)
Primitive types	• The type itself, such as `double` • **rti::flat::PrimitiveOffset** (p. 2134) (when the member is optional)

Offsets for user types are generated by **rtiddsgen** and provide methods to access the type's members by their names. Offsets for arrays and sequences provide methods to access each element.

Some offsets allow accessing the data through a pointer to the equivalent plain C++ type, which generally provides better performance. See **rti::flat::plain_cast()** (p. 773).

7.226.2 Offset Error Management

Functions that return an Offset may return a "null" Offset (one such that **is_null()** (p. 2128) returns `true`).

An Offset may be null if a member doesn't exist in the **Sample** (p. 769). For example, if the member 'my_final' in **My↩ FlatMutable** (p. 770) doesn't exist (because it wasn't added while building the Sample, or because it wasn't received in the subscribing application), **MyFlatMutableOffset::my_final()** (p. 2030) returns an null Offset. Note that a **member in a final type** (for example, **MyFlatFinalOffset::my_complex()** (p. 2017)) always exists, except in the case of an error.

In addition to that, a function may return a null Offset for any error condition.

7.226.3 Function Documentation

7.226.3.1 plain_cast() [1/2]

```
template<typename OffsetType >
flat_type_traits< OffsetType >::plain_type * rti::flat::plain_cast (
            OffsetType & offset )
```

Casts into an equivalent plain C++ type.

Some FlatData types can be cast to their equivalent **plain** definition as a regular non-FlatData C++ type. This is a more efficient way to access the data. This function casts, if possible, the member pointed to by the offset argument to an equivalent plain C++ type. Any changes made through the plain type are reflected in the FlatData sample.

Template Parameters

OffsetType	The Offset type

Precondition

plain_cast requires the type to meet the following restrictions:

- The type must be a final struct, or an array or sequence of members of a type that meets these restrictions (including primitive types)
- The type must be defined in a way such that the packing of the plain C++ type doesn't differ from the padding in XCDR2.
- The type may not inherit from another type.
- In addition, the sample must be serialized in the native endianness. For example, if a subscribing application on a little-endian platform receives a sample published from a big-endian system, it is not possible to plain↩ _cast the sample or any of its members, except if the member is a primitive array or sequence of 1-byte elements.

offset.is_cpp_compatible() (p. 2128) indicates if the member meets the requirements. If the type doesn't, this function returns NULL.

Parameters

offset	The offset to the member to cast.

Returns

The data that `offset` referred to, cast as a plain C++ type with the same definition. If `offset` is an array or sequence member, the returned pointer represents a C++ array with the same number of elements.

The following table summarizes the possible parameters to this function (assuming that they meet the previous requirements), and the return type in each case:

OffsetType	return type
Offset to a final struct, such as **MyFlatFinalOffset** (p. 2014)	`MyFlatFinalPlainHelper*` (pointer to a single element). This is a type with the same IDL definition, but without `@language_binding(FLAT_DATA)`.

OffsetType	return type
Offset to an array of final structs, such as **Final↩ ArrayOffset** (p. 1911)<**MyFlatFinal↩ Offset** (p. 2014), N>	MyFlatFinalPlainHelper* (array of N elements)
Offset to a sequence of final structs, such as **SequenceOffset** (p. 2204)<**MyFlatFinal↩ Offset** (p. 2014)>	MyFlatFinalPlainHelper* (array of offset.element_count() elements)
Offset to an array of primitive types, such as **PrimitiveArrayOffset** (p. 2132)<int32↩ _t, N>	int32_t* (array of N elements)
Offset to a sequence of primitive types, such as **PrimitiveSequenceOffset** (p. 2138)<int32_t>	int32_t* (array of offset.element_count() elements)

Due to the performance advantages that plain_cast offers, it is recommended to define FlatData types in a way that their largest member(s) can be plain_cast.

The following example shows how to use plain_cast to cast a **MyFlatFinalOffset** (p. 2014) into the type with the same IDL definition as **MyFlatFinal** (p. 770), but without the @language_binding(FLAT_DATA) annotation:

```
MyFlatMutable *my_sample = ...;
auto my_root = my_sample->root();
auto my_final = my_root.my_final();
// Get the plain C++ type and modify an array
auto my_final_plain = rti::flat::plain_cast(my_final);
my_final_plain->my_primitive(3); // this is using the plain C++ setter
my_final_plain->my_complex_array()[1].x(20); // this is a std::array
// The change to my_complex_array is reflected in the FlatData sample
std::cout « my_final.my_primitive() « std::endl; // 3
std::cout « my_final.my_complex_array().get_element(1).x() « std::endl; // 20
```

This example shows how to use plain_cast to efficiently build a sequence of final elements and a sequence of integers, both members of a mutable type:

```
MyFlatMutableBuilder builder = rti::flat::build_data(writer);
// 1)
//
// Build a sequence of 5 elements with add_n() instead of 5 calls to
// add_next()
rti::flat::FinalSequenceBuilder<MyFlatFinalOffset> seq_builder =
          builder.build_my_final_seq();
seq_builder.add_n(5);
// Finish the member, getting the offset to the member
rti::flat::SequenceOffset<MyFlatFinalOffset> seq_offset = seq_builder.finish();
// Cast it to an array of plain C++ type and initialize it.
// This way to initialize it is more efficient than add_next().
auto seq_elements = rti::flat::plain_cast(seq_offset);
for (int i = 0; i < 5; i++) {
    seq_elements[i].my_primitive(i);
    // ...
}
// 2)
//
// Build a sequence of 1000 integers
auto seq_builder = builder.build_my_primitive_seq();
// make space for 1000 elements, but leave them uninitialized
seq_builder.add_n(1000);
auto seq_offset = seq_builder.finish();
// Initialize the elements:
int32_t *elements = rti::flat::plain_cast(seq_offset);
for (int i = 0; i < 1000; i++) {
    elements[i] = ...;
}
// ... continue building the sample using 'builder'
```

7.226.3.2 plain_cast() [2/2]

```
template<typename OffsetType >
const  flat_type_traits< OffsetType >::plain_type * rti::flat::plain_cast (
             const OffsetType & offset )
```

Const version of plain_cast.

7.227 FlatData Topic-Types

<<***extension***>> *(p. 432)* FlatData Language Binding for IDL topic-types

Modules

- **FlatData Builders**

  A Builder allows creating and initializing variable-size data.
- **FlatData Samples**

  A Sample represents an instance of the IDL topic-type and contains the data in serialized format.
- **FlatData Offsets**

  Offsets provide access to the members of a FlatData Sample.

7.227.1 Detailed Description

<<***extension***>> *(p. 432)* FlatData Language Binding for IDL topic-types

Note

For a complete description of the FlatData language binding and its benefits, and a tutorial, see the "Sending Large Data" chapter in the **RTI Connext User's Manual**.

For buildable **code examples**, see `https://community.rti.com/kb/flatdata-and-zerocopy-examples`.

The FlatData language binding is available in the Traditional C++ API and in the Modern C++ API.

RTI FlatData™ is a language binding for IDL types in which the in-memory representation of a sample matches the wire representation. Therefore, the cost of serialization/deserialization is zero.

To select FlatData as the language binding of a type, annotate it with `@language_binding(FLAT_DATA)` in IDL or apply the attribute `languageBinding="flat_data"` in XML.

There are some restrictions regarding the kinds of types to which the FlatData language binding can be applied.

- For final types, the FlatData language binding can be applied only to fixed-size types. A fixed-size type is a type whose wire representation always has the same size. This includes primitive types, arrays of fixed-size types, and structs containing only members of fixed-size types. Unions are not fixed-size types.
- For mutable types, any member is permitted.

- Extensible types cannot be marked as FlatData.

Final types are more efficient, but more restrictive. In general, a good compromise is to define mutable top-level types, but making sure their largest data members are final.

When a type is marked with the FlatData language binding, its mapping into C++ is different than that of a regular type (plain language binding). Rather than a single C++ class with direct access to its members, for a FlatData type **rtiddsgen** generates the following:

- A **Sample** (p. 769), the data in serialized format
- An **Offset** (p. 771), which allows reading the data members of that type inside a Sample, and modifying them without changing the size
- A **Builder** (p. 766), if the type is mutable, which allows creating a variable-size Sample

For example, for the IDL types **MyFlatFinal** (p. 770) and **MyFlatMutable** (p. 770) **rtiddsgen** generates the following C++ types:

- The Sample types **MyFlatFinal** (p. 770) and **MyFlatMutable** (p. 770)
- The Offset types **MyFlatFinalOffset** (p. 2014) and **MyFlatMutableOffset** (p. 2025).
- The Builder type **MyFlatMutableBuilder** (p. 2018).

7.227.2 Publishing FlatData

The typical way to publish FlatData samples includes the following steps:

- Create a **FooDataWriter** (p. 1944) as you would for a non-FlatData (plain) topic-type (see **Setting up a data writer** (p. 400)).

(For final topic-types such as **MyFlatFinal** (p. 770))

- Obtain a FlatData sample with **FooDataWriter::get_loan** (p. 1963).
```
MyFlatFinal *foo_sample = writer.get_loan();
```
- Initialize the sample, starting from the **root()** (p. 2189) offset.
```
MyFlatFinalOffset foo_root = foo_sample->root();
foo_root.my_primitive(3);
auto my_complex_offset = foo_root.my_complex();
// ... initialize my_complex_offset
```

(For mutable topic-types such as **MyFlatMutable** (p. 770))

- Obtain a **Builder** (p. 766) to create a variable-size sample with **rti::flat::build_data()** (p. 768).
```
MyFlatMutableBuilder foo_builder = rti::flat::build_data(writer);
```

- Use this builder (and possibly nested member builders) to initialize the members.
```
foo_builder.add_my_primitive(3);
auto my_mutable_builder = foo_builder.build_my_mutable();
// ... build 'my_mutable' (a member with a mutable type)
my_mutable_builder.finish(); // completes a member
auto my_final_offset = foo_builder.add_my_final();
// ... initialize 'my_final' (a member with a final type)
```
- Obtain the completed sample with **MyFlatMutableBuilder::finish_sample()** (p. 2021). After that, the builder is no longer usable, and the sample size cannot change.
```
MyFlatMutable *foo_sample = foo_builder.finish_sample();
```
- Optionally, it is possible to change the values of the sample accessing its root(), as long as the size doesn't change. For example, if the sample was built with a sequence member with two elements, it is possible to modify any of those elements, but it's not possible to add a third element.

(For both final and mutable topic-types)

- Write the sample with **FooDataWriter::write** (p. 1951).
```
writer.write(*foo_sample);
```

After write, the DataWriter owns the FlatData sample. This allows avoiding additional copies, the main goal of the Flat↩ Data language binding. This means that the sample cannot be reused. The DataWriter will return it to the sample pool when appropriate, as described in **FooDataWriter::get_loan** (p. 1963).

7.227.3 Subscribing to FlatData

To subscribe to a topic using a FlatData topic-type:

- Create a **FooDataReader** (p. 1917) normally (see **Setting up a data reader** (p. 403)).
- Read or take the samples using a loaning operation, such as **FooDataReader::take** (p. 1920) (copying read/take operations cannot be used with FlatData types).
- Access the **root()** (p. 2189) offset, and any of the sample's members from there.

Note that the language binding is a local concept. It is possible to publish with a FlatData topic-type and subscribe to it with a **plain** topic-type with the same (or assignable) definition. It is also possible to use a plain topic-type on the publisher side and subscribe to it using FlatData. The DataWriter or DataReader of the plain topic-type has to use **DDS_XCDR2_DATA_REPRESENTATION** (p. 569) in **DDS_DataRepresentationQosPolicy** (p. 886).

Chapter 8

Namespace Documentation

8.1 connext Namespace Reference

Namespace for Connext.

Classes

- class **AlreadyDeletedException**

  The object target of this operation has already been deleted.
- class **BadParameterException**

  Illegal parameter value.
- class **IllegalOperationException**

  The operation was called under improper circumstances.
- class **ImmutablePolicyException**

  Application attempted to modify an immutable QoS policy.
- class **InconsistentPolicyException**

  Application specified a set of QoS policies that are not consistent with each other.
- class **IsInvalidSamplePredicate**

  Predicate-class to determine if a sample contains invalid data.
- class **IsReplyRelatedPredicate**

  Predicate-class to match replies by their related request.
- class **IsValidSamplePredicate**

  Predicate-class to determine if a sample contains valid data.
- class **LoanedSamples**

  Provides access to a collection of middleware-loaned samples.
- class **LogicException**

  Base class of all RTI Connext exceptions caused by a logic error.
- class **MessagingLibraryVersion**

  The Connext Messaging Library version.
- class **MessagingVersion**

  The Connext Messaging version.

- class **NotEnabledException**

  *Operation invoked on a **DDSEntity** (p. 1731) that is not yet enabled.*
- class **OutOfResourcesException**

  RTI Connext ran out of the resources needed to complete the operation.
- class **PreconditionNotMetException**

  A pre-condition for the operation was not met.
- class **Replier**

  Allows receiving requests and sending replies.
- class **ReplierListener**

  *Called when a **connext::Replier** (p. 2139) has new available requests.*
- class **ReplierParams**

  *Contains the parameters for creating a **connext::Replier** (p. 2139).*
- class **Requester**

  Allows sending requests and receiving replies.
- class **RequesterParams**

  *Contains the parameters for creating a **connext::Requester** (p. 2157).*
- class **RuntimeException**

  Generic, unspecified error.
- class **Sample**

  A data sample and related info received from the middleware.
- class **SampleIterator**

  STL-compliant random-access iterator for SampleRef<T>
- class **SampleRef**

  A data sample and related information received from the middleware.
- class **SimpleReplier**

  A callback-based replier.
- class **SimpleReplierListener**

  *The listener called by a **SimpleReplier** (p. 2206).*
- class **SimpleReplierParams**

  *Contains the parameters for creating a **connext::SimpleReplier** (p. 2206).*
- class **TimeoutException**

  The operation timed out (does not apply to wait or receive operations)
- class **UnsupportedException**

  Unsupported operation.
- class **WriteSample**

  A sample for writing data.
- class **WriteSampleRef**

  A reference to a data sample for writing.

Functions

- template<typename T , bool IsConst>
  ValidSampleIterator< T, IsConst > **make_valid_sample_iterator** (**SampleIterator**< T, IsConst > related_↩ iterator)

  Creates an iterator that only returns valid samples.
- template<typename T >
  LoanedSamples< T > **move** (**LoanedSamples**< T > &ls) throw ()

  *Creates a new **LoanedSamples** (p. 1995) instance by moving the contents of an existing one.*

8.1.1 Detailed Description

Namespace for Connext.

8.2 rti Namespace Reference

The RTI namespace.

Namespaces

- namespace **flat**

  <<***extension***>> *(*p. *432) Support for* ***FlatData Topic-Types*** *(*p. *776)*

8.2.1 Detailed Description

The RTI namespace.

8.3 rti::flat Namespace Reference

<<***extension***>> *(*p. *432)* Support for **FlatData Topic-Types** (p. 776)

Classes

- class **AbstractAlignedList**

  Base class of Offsets to sequences and arrays of non-primitive members.
- class **AbstractBuilder**

  Base class of all ***Builders*** *(*p. *766).*
- class **AbstractListBuilder**

  Base class of all array and sequence builders.
- class **AbstractPrimitiveList**

  Base class for Offsets to sequences and arrays of primitive types.
- class **AbstractSequenceBuilder**

  Base class of Builders for sequence members.
- class **AggregationBuilder**

  Base class of struct and union builders.
- class **FinalAlignedArrayOffset**

  Offset to an array of final elements.
- class **FinalArrayOffset**

  Offset to an array of final elements.
- class **FinalOffset**

  The base class of all Offsets to a final struct type.

- class **FinalSequenceBuilder**

  Builds a sequence member of fixed-size elements.
- struct **flat_type_traits**

  *Given a **Sample** (p. 2187), an Offset or a Builder, it allows obtaining the other types.*
- class **MutableArrayBuilder**

  Builds an array member of variable-size elements.
- class **MutableArrayOffset**

  Offset to an array of variable-size elements.
- class **MutableOffset**

  The base class of all Offsets to a final struct type.
- class **MutableSequenceBuilder**

  Builds a sequence member of variable-size elements.
- class **OffsetBase**

  Base class of all Offset types.
- class **PrimitiveArrayOffset**

  Offset to an array of primitive elements.
- struct **PrimitiveConstOffset**

  A const Offset to an optional primitive member.
- struct **PrimitiveOffset**

  An Offset to an optional primitive member.
- class **PrimitiveSequenceBuilder**

  Builds a sequence of primitive members.
- class **PrimitiveSequenceOffset**

  Offset to a sequence of primitive elements.
- class **Sample**

  The generic definition of FlatData topic-types.
- class **SequenceIterator**

  Iterator for collections of Offsets.
- class **SequenceOffset**

  Offset to a sequence of non-primitive elements.
- class **StringBuilder**

  Builds a string.
- class **StringOffset**

  Offset to a string.
- class **UnionBuilder**

  Base class of builders for user-defined mutable unions.

Functions

- template<typename OffsetType >
  flat_type_traits< OffsetType >::plain_type * **plain_cast** (OffsetType &offset)

  Casts into an equivalent plain C++ type.
- template<typename OffsetType >
  const **flat_type_traits**< OffsetType >::plain_type * **plain_cast** (const OffsetType &offset)

  Const version of plain_cast.
- template<typename TopicType >
  rti::flat::flat_type_traits< TopicType >::builder **build_data** (typename TopicType::DataWriter *writer)

Begins building a new sample.

- template<typename BuilderType >
  void **discard_builder** (typename **rti::flat::flat_type_traits**< BuilderType >::flat_type::DataWriter *writer, BuilderType &builder)

  Discards a sample builder.

8.3.1 Detailed Description

<<***extension***>> *(*p. *432)* Support for **FlatData Topic-Types** (p. 776)

Chapter 9

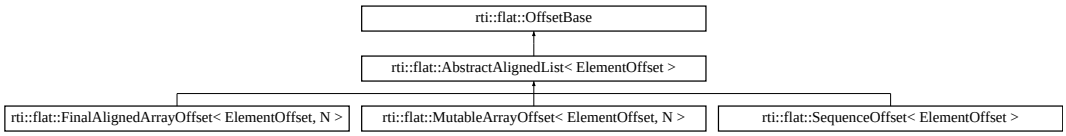
Class Documentation

9.1 rti::flat::AbstractAlignedList< ElementOffset > Class Template Reference

Base class of Offsets to sequences and arrays of non-primitive members.

`#include <SequenceOffsets.hpp>`

Inheritance diagram for rti::flat::AbstractAlignedList< ElementOffset >:



Public Types

- typedef **SequenceIterator**< ElementOffset, typename ElementOffset::offset_kind > **iterator**
  *The iterator type, **SequenceIterator** (p. 2198).*

Public Member Functions

- **iterator begin** ()
  Gets an iterator to the first Offset.
- **iterator end** ()
  Gets an iterator to the past-the-end element.

9.1.1 Detailed Description

template<typename ElementOffset>
class rti::flat::AbstractAlignedList< ElementOffset >

Base class of Offsets to sequences and arrays of non-primitive members.

Template Parameters

ElementOffset	The Offset type of the elements

9.1.2 Member Typedef Documentation

9.1.2.1 iterator

```
template<typename ElementOffset >
typedef SequenceIterator<ElementOffset, typename ElementOffset::offset_kind> rti::flat::↩
AbstractAlignedList< ElementOffset >::iterator
```

The iterator type, **SequenceIterator** (p. 2198).

9.1.3 Member Function Documentation

9.1.3.1 begin()

```
template<typename ElementOffset >
iterator rti::flat::AbstractAlignedList< ElementOffset >::begin ( ) [inline]
```

Gets an iterator to the first Offset.

begin() (p. 786) and **end()** (p. 786) enable the use of range-for loops, for example:

```
SequenceOffset<MyFlatMutableOffset> sequence_offset = my_type_offset.my_sequence();
for (auto element : sequence_offset) {
    std::cout « element.x() « std::endl;
}
```

References **rti::flat::OffsetBase::get_buffer_size()**.

9.1.3.2 end()

```
template<typename ElementOffset >
iterator rti::flat::AbstractAlignedList< ElementOffset >::end ( ) [inline]
```

Gets an iterator to the past-the-end element.

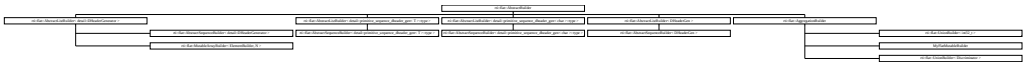
References **rti::flat::OffsetBase::get_buffer_size()**.

9.2 rti::flat::AbstractBuilder Class Reference

Base class of all **Builders** (p. 766).

`#include <Builder.hpp>`

Inheritance diagram for rti::flat::AbstractBuilder:



Public Member Functions

- void **discard** ()
  Discards a member in process of being built.
- bool **is_nested** () const
  Returns whether this is a member Builder.
- bool **is_valid** () const
  Whether this Builder is valid.
- rti::xcdr::length_t **capacity** () const
  Returns the total capacity in bytes.
- bool **check_failure** ()
  Checks if the previous operation failed and resets the failure flag.

Protected Member Functions

- virtual ~**AbstractBuilder** ()
  If this is a member Builder, it calls finish().

9.2.1 Detailed Description

Base class of all **Builders** (p. 766).

9.2.2 Constructor & Destructor Documentation

9.2.2.1 ~AbstractBuilder()

`virtual rti::flat::AbstractBuilder::~AbstractBuilder ( ) [inline], [protected], [virtual]`

If this is a member Builder, it calls finish().

If this Builder is building a member (that is, **is_nested()** (p. 788) is true), and the object goes out of scope before finish() has been called, its destructor calls finish(). Note, however, that it won't report any error.

If this Builder is building a sample (!is_nested()), its destructor doesn't do anything.

9.2.3 Member Function Documentation

9.2.3.1 discard()

```
void rti::flat::AbstractBuilder::discard ( )  [inline]
```

Discards a member in process of being built.

This function ends the creation of a member, returning the Builder of the type that contains the member to its previous state, as if this member had never been built.

Precondition

This object must be a member Builder, not a sample Builder.

This method is useful when during the building of a member an error occurs and the application wants to roll back, instead of finishing an incomplete member.

9.2.3.2 is_nested()

```
bool rti::flat::AbstractBuilder::is_nested ( ) const  [inline]
```

Returns whether this is a member Builder.

A member Builder is a Builder that has been created by calling a "`build_<member>`" function on another Builder (for example, **MyFlatMutableBuilder::build_my_mutable()** (p. 2024)).

Returns

True if this is a member Builder, or false if this is a sample Builder.

9.2.3.3 is_valid()

```
bool rti::flat::AbstractBuilder::is_valid ( ) const  [inline]
```

Whether this Builder is valid.

A Builder is not valid when it is default-constructed, or after any of these functions is called: finish(), finish_sample(), **discard()** (p. 788).

Since the Traditional C++ API doesn't use exceptions, certain function failures due to the Builder running out resources are also reported by invalidating the Builder.

9.2.3.4 capacity()

```
rti::xcdr::length_t rti::flat::AbstractBuilder::capacity ( ) const  [inline]
```

Returns the total capacity in bytes.

The capacity is the total number of bytes this Builder can contain. For a sample Builder (that is, one such that **is_nested()** (p. 788) returns false), **rti::flat::build_data** (p. 768) reserves enough bytes to accommodate any sample of a given type.

9.2.3.5 check_failure()

```
bool rti::flat::AbstractBuilder::check_failure ( )  [inline]
```

Checks if the previous operation failed and resets the failure flag.

This function must be called after each Builder operation to check if it succeeded. Note that after calling **check_failure()** (p. 789) the error flag is reset, so a subsequent call will always return false.

Returns

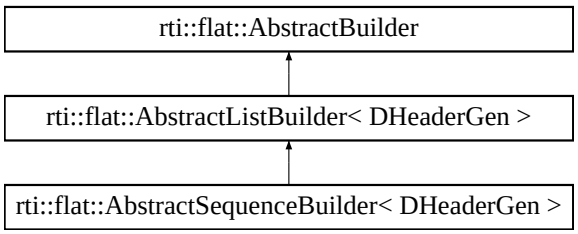
True if there was a failure in the previous operation, false if there was no failure since the last call to check_failure.

9.3 rti::flat::AbstractListBuilder< DHeaderGen > Class Template Reference

Base class of all array and sequence builders.

```
#include <SequenceBuilders.hpp>
```

Inheritance diagram for rti::flat::AbstractListBuilder< DHeaderGen >:



Protected Member Functions

- unsigned int **element_count** () const

  Returns the current number of elements that have been added.

Additional Inherited Members

9.3.1 Detailed Description

template<typename DHeaderGen>
class rti::flat::AbstractListBuilder< DHeaderGen >

Base class of all array and sequence builders.

9.3.2 Member Function Documentation

9.3.2.1 element_count()

```
template<typename DHeaderGen >
unsigned int  rti::flat::AbstractListBuilder< DHeaderGen >::element_count ( ) const  [inline],
[protected]
```

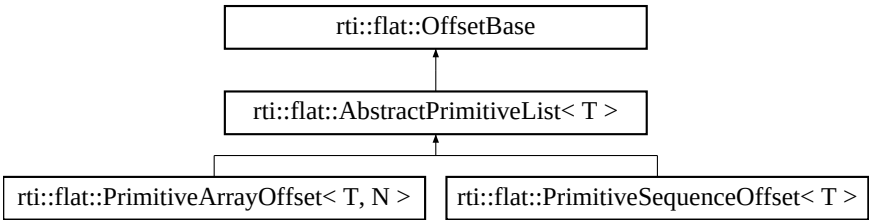
Returns the current number of elements that have been added.

9.4 rti::flat::AbstractPrimitiveList< T > Class Template Reference

Base class for Offsets to sequences and arrays of primitive types.

`#include <SequenceOffsets.hpp>`

Inheritance diagram for rti::flat::AbstractPrimitiveList< T >:



Public Member Functions

- T **get_element** (unsigned int i) const
  Returns an element by index.
- bool **set_element** (unsigned int i, T value)
  Sets an element by index.

9.4.1 Detailed Description

template<typename T>
class rti::flat::AbstractPrimitiveList< T >

Base class for Offsets to sequences and arrays of primitive types.

Template Parameters

T	The primitive type

9.4.2 Member Function Documentation

9.4.2.1 get_element()

```
template<typename T >
T  rti::flat::AbstractPrimitiveList< T >::get_element (
            unsigned int i ) const  [inline]
```

Returns an element by index.

Parameters

i	The zero-based index of the element

See also

rti::flat::plain_cast() (p. 773) for a method to access all the elements at once

9.4.2.2 set_element()

```
template<typename T >
bool  rti::flat::AbstractPrimitiveList< T >::set_element (
            unsigned int i,
            T value )  [inline]
```

Sets an element by index.

Parameters

i	The zero-based index of the element to set
value	The value to set

Returns

true if it was possible to set the element, or false if this collection has less than i - 1 elements.

See also

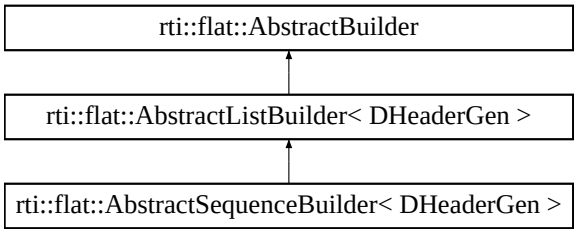
rti::flat::plain_cast() (p. 773) for a method to access all the elements at once

9.5 rti::flat::AbstractSequenceBuilder< DHeaderGen > Class Template Reference

Base class of Builders for sequence members.

`#include <SequenceBuilders.hpp>`

Inheritance diagram for rti::flat::AbstractSequenceBuilder< DHeaderGen >:



Public Member Functions

- unsigned int **element_count** () const

  Returns the current number of elements that have been added.

Additional Inherited Members

9.5.1 Detailed Description

template<typename DHeaderGen>
class rti::flat::AbstractSequenceBuilder< DHeaderGen >

Base class of Builders for sequence members.

This class contains only implementation details and doesn't add any public function to **AbstractListBuilder** (p. 789).

9.5.2 Member Function Documentation

9.5.2.1 element_count()

```
template<typename DHeaderGen >
unsigned int  rti::flat::AbstractListBuilder< DHeaderGen >::element_count ( ) const  [inline]
```

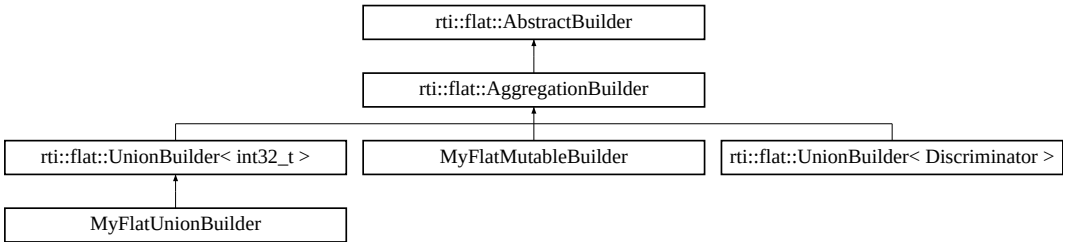
Returns the current number of elements that have been added.

9.6 rti::flat::AggregationBuilder Class Reference

Base class of struct and union builders.

```
#include <AggregationBuilders.hpp>
```

Inheritance diagram for rti::flat::AggregationBuilder:



Additional Inherited Members

9.6.1 Detailed Description

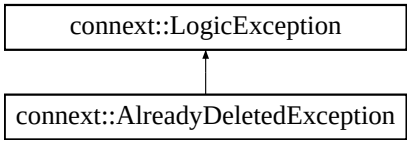
Base class of struct and union builders.

This class contains implementation details and doesn't add any public function to **AbstractBuilder** (p. 787). See **My↩ FlatMutableBuilder** (p. 2018) for a concrete example of a struct builder.

9.7 connext::AlreadyDeletedException Class Reference

The object target of this operation has already been deleted.

Inheritance diagram for connext::AlreadyDeletedException:



9.7.1 Detailed Description

The object target of this operation has already been deleted.

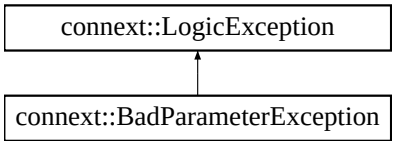
See also

DDS_RETCODE_ALREADY_DELETED (p. 533)

9.8 connext::BadParameterException Class Reference

Illegal parameter value.

Inheritance diagram for connext::BadParameterException:



9.8.1 Detailed Description

Illegal parameter value.

See also

DDS_RETCODE_BAD_PARAMETER (p. 532)

9.9 DDS_AcknowledgmentInfo Struct Reference

Information about an application-level acknowledged sample.

Public Attributes

- **DDS_InstanceHandle_t subscription_handle**

  *Subscription handle of the acknowledging **DDSDataReader** (p. 1556).*
- struct **DDS_SampleIdentity_t sample_identity**

  Identity of the sample being acknowledged.
- **DDS_Boolean valid_response_data**

  Flag indicating validity of the user response data in the acknowledgment.
- struct **DDS_AckResponseData_t response_data**

  User data payload of application-level acknowledgment message.

9.9.1 Detailed Description

Information about an application-level acknowledged sample.

When acknowledging a sample, the reader provides the writer with information about the sample being acknowledged. The AcknowledgmentInfo structure provides the identity and cookie of the sample being acknowledged, as well as user data payload provided by the reader.

9.9.2 Member Data Documentation

9.9.2.1 subscription_handle

```
DDS_InstanceHandle_t DDS_AcknowledgmentInfo::subscription_handle
```

Subscription handle of the acknowledging **DDSDataReader** (p. 1556).

9.9.2.2 sample_identity

```
struct DDS_SampleIdentity_t DDS_AcknowledgmentInfo::sample_identity
```

Identity of the sample being acknowledged.

See also

DDS_SampleIdentity_t (p. 1343)

9.9.2.3 valid_response_data

```
DDS_Boolean DDS_AcknowledgmentInfo::valid_response_data
```

Flag indicating validity of the user response data in the acknowledgment.

This flag is true when the **DDS_RtpsReliableReaderProtocol_t::min_app_ack_response_keep_duration** (p. 1318) has not yet elapsed for the acknowledgment's response data.

The flag is false when that duration has elapsed for the response data.

9.9.2.4 response_data

`struct` **`DDS_AckResponseData_t`** `DDS_AcknowledgmentInfo::response_data`

User data payload of application-level acknowledgment message.

Response data set by **DDSDataReader** (p. 1556) when sample was acknowledged.

9.10 DDS_AckResponseData_t Struct Reference

Data payload of an application-level acknowledgment.

Public Attributes

- struct **DDS_OctetSeq value**
  a sequence of octets

9.10.1 Detailed Description

Data payload of an application-level acknowledgment.

9.10.2 Member Data Documentation

9.10.2.1 value

`struct` **`DDS_OctetSeq`** `DDS_AckResponseData_t::value`

a sequence of octets

[default] empty (zero-length)

[range] Octet sequence of length [0, **DDS_DataReaderResourceLimitsQosPolicy::max_app_ack_response_length** (p. 881)],

9.11 DDS_AllocationSettings_t Struct Reference

Resource allocation settings.

Public Attributes

- **DDS_Long initial_count**
  The initial count of resources.
- **DDS_Long max_count**
  The maximum count of resources.
- **DDS_Long incremental_count**
  The incremental count of resources.

9.11.1 Detailed Description

Resource allocation settings.

QoS:

DDS_DomainParticipantResourceLimitsQosPolicy (p. 977)

9.11.2 Member Data Documentation

9.11.2.1 initial_count

```
DDS_Long DDS_AllocationSettings_t::initial_count
```

The initial count of resources.

The initial resources to be allocated.

[default] It depends on the case.

[range] [0, 1 million], <= max_count

9.11.2.2 max_count

```
DDS_Long DDS_AllocationSettings_t::max_count
```

The maximum count of resources.

The maximum resources to be allocated.

[default] Depends on the case.

[range] [1, 1 million] or **DDS_LENGTH_UNLIMITED** (p. 636), >= initial_count

9.11.2.3 incremental_count

```
DDS_Long DDS_AllocationSettings_t::incremental_count
```

The incremental count of resources.

The resource to be allocated when more resources are needed.

[default] Depends on the case.

[range] -1 (Double the amount of extra memory allocated each time memory is needed) or [1,1 million] (or = 0 only if initial_count == max_count)

9.12 DDS_AnnotationParameterValue Struct Reference

Annotation parameter value.

9.12.1 Detailed Description

Annotation parameter value.

This structure is use to represent an annotation parameter value. For example, the value of the annotation **[default]** for an aggregation type member.

9.13 DDS_AsynchronousPublisherQosPolicy Struct Reference

<<***extension***>> *(*p. *213)* Configures the mechanism that sends user data in an external middleware thread.

Public Attributes

- **DDS_Boolean disable_asynchronous_write**

  [DEPRECATED] (p. ***384)*** <<***extension***>> *(*p. *213) Disable asynchronous publishing.*
- struct **DDS_ThreadSettings_t thread**

  <<***extension***>> *(*p. *213) Settings of the publishing thread.*
- **DDS_Boolean disable_asynchronous_batch**

  [DEPRECATED] (p. ***384)*** <<***extension***>> *(*p. *213) Disable asynchronous batch flushing.*
- struct **DDS_ThreadSettings_t asynchronous_batch_thread**

  <<***extension***>> *(*p. *213) Settings of the batch flushing thread.*
- **DDS_Boolean disable_topic_query_publication**

  [DEPRECATED] (p. ***384)*** <<***extension***>> *(*p. *213) Disable topic query publication.*
- struct **DDS_ThreadSettings_t topic_query_publication_thread**

  <<***extension***>> *(*p. *213) Settings of the* ***DDSTopicQuery*** *(*p. *1895) publication thread.*

9.13.1 Detailed Description

<<***extension***>> *(p. 213)* Configures the mechanism that sends user data in an external middleware thread.

Specifies the asynchronous publishing and asynchronous batch flushing settings of the **DDSPublisher** (p. 1818) instances.

The QoS policy specifies whether asynchronous publishing and asynchronous batch flushing are enabled for the **DDSDataWriter** (p. 1589) entities belonging to this **DDSPublisher** (p. 1818). If so, the publisher will spawn up to two threads, one for asynchronous publishing and one for asynchronous batch flushing.

This policy also configures the settings of the **DDSTopicQuery** (p. 1895) publication thread. The publisher will spawn this thread only if one or more DataWriters enable TopicQueries.

See also

DDS_BatchQosPolicy (p. 810).

DDS_PublishModeQosPolicy (p. 1279).

Entity:

DDSPublisher (p. 1818)

Properties:

RxO (p. 556) = N/A

Changeable (p. 556) = **NO** (p. 556)

9.13.2 Usage

You can use this QoS policy to reduce the amount of time your application thread spends sending data.

The asynchronous *publisher* thread is shared by all **DDS_ASYNCHRONOUS_PUBLISH_MODE_QOS** (p. 629) **DDSDataWriter** (p. 1589) instances that belong to this publisher and handles their data transmission chores.

The asynchronous *batch flushing* thread is shared by all **DDSDataWriter** (p. 1589) instances with batching enabled that belong to this publisher.

This QoS policy also allows you to adjust the settings of the asynchronous publishing and the asynchronous batch flushing threads. To use different threads for two different **DDSDataWriter** (p. 1589) entities, the instances must belong to different **DDSPublisher** (p. 1818) instances.

A **DDSPublisher** (p. 1818) must have asynchronous publishing enabled for its **DDSDataWriter** (p. 1589) instances to write asynchronously.

A **DDSPublisher** (p. 1818) must have asynchronous batch flushing enabled in order to flush the batches of its **DDSDataWriter** (p. 1589) instances asynchronously. However, no asynchronous batch flushing thread will be started until the first **DDSDataWriter** (p. 1589) instance with batching enabled is created from this **DDSPublisher** (p. 1818).

9.13.3 Member Data Documentation

9.13.3.1 disable_asynchronous_write

```
DDS_Boolean DDS_AsynchronousPublisherQosPolicy::disable_asynchronous_write
```

[DEPRECATED] (p. **384)** <<***extension***>> *(*p. *213)* Disable asynchronous publishing.

If set to **DDS_BOOLEAN_TRUE** (p. 513), any **DDSDataWriter** (p. 1589) created with **DDS_ASYNCHRONOUS_↩ PUBLISH_MODE_QOS** (p. 629) will fail with **DDS_RETCODE_INCONSISTENT_POLICY** (p. 533).

[default] DDS_BOOLEAN_FALSE (p. 513)

Categories

Immutable (p. 333), **Flow Control** (p. 282), **Concurrency** (p. 258)

9.13.3.2 thread

```
struct DDS_ThreadSettings_t DDS_AsynchronousPublisherQosPolicy::thread
```

<<***extension***>> *(*p. *213)* Settings of the publishing thread.

There is only one asynchronous publishing thread per **DDSPublisher** (p. 1818).

[default] priority below normal.
The actual value depends on your architecture:

For Windows: -2
For Linux: OS default priority
For a complete list of platform specific values, please refer to `Platform Notes`.

[default] The actual value depends on your architecture:

For Windows: OS default stack size
For Linux: OS default stack size
For a complete list of platform specific values, please refer to `Platform Notes`.

[default] mask = **DDS_THREAD_SETTINGS_KIND_MASK_DEFAULT** (p. 547)

Categories

Immutable (p. 333), **Concurrency** (p. 258)

9.13.3.3 disable_asynchronous_batch

DDS_Boolean DDS_AsynchronousPublisherQosPolicy::disable_asynchronous_batch

[DEPRECATED] (p. **384)** <<***extension***>> *(*p. *213)* Disable asynchronous batch flushing.

If set to **DDS_BOOLEAN_TRUE** (p. 513), any **DDSDataWriter** (p. 1589) created with batching enabled will fail with **DDS_RETCODE_INCONSISTENT_POLICY** (p. 533).

If **DDS_BatchQosPolicy::max_flush_delay** (p. 812) is different than **DDS_DURATION_INFINITE** (p. 522), **DDS_↩ AsynchronousPublisherQosPolicy::disable_asynchronous_batch** (p. 800) must be set **DDS_BOOLEAN_FALSE** (p. 513).

[default] DDS_BOOLEAN_FALSE (p. 513)

Categories

Immutable (p. 333), **Concurrency** (p. 258), **Batching** (p. 257)

9.13.3.4 asynchronous_batch_thread

struct **DDS_ThreadSettings_t** DDS_AsynchronousPublisherQosPolicy::asynchronous_batch_thread

<<***extension***>> *(*p. *213)* Settings of the batch flushing thread.

There is only one asynchronous batch flushing thread per **DDSPublisher** (p. 1818).

[default] priority below normal.
The actual value depends on your architecture:

For Windows: -2
For Linux: OS default priority
For a complete list of platform specific values, please refer to `Platform Notes`. **[default]** The actual value depends on your architecture:

For Windows: OS default stack size
For Linux: OS default stack size
For a complete list of platform specific values, please refer to `Platform Notes`.

[default] mask = **DDS_THREAD_SETTINGS_KIND_MASK_DEFAULT** (p. 547)

Categories

Immutable (p. 333), **Concurrency** (p. 258), **Batching** (p. 257)

9.13.3.5 disable_topic_query_publication

```
DDS_Boolean DDS_AsynchronousPublisherQosPolicy::disable_topic_query_publication
```

[DEPRECATED] (p. **384**) <<***extension***>> *(p. 213)* Disable topic query publication.

If set to **DDS_BOOLEAN_TRUE** (p. 513), any **DDSDataWriter** (p. 1589) created with **DDS_TopicQueryDispatch↩ QosPolicy::enable** (p. 1405) set to **DDS_BOOLEAN_TRUE** (p. 513) will fail with **DDS_RETCODE_INCONSISTENT↩ _POLICY** (p. 533).

[default] DDS_BOOLEAN_FALSE (p. 513)

Categories

Immutable (p. 333), **Concurrency** (p. 258), **Topic Queries** (p. 330)

9.13.3.6 topic_query_publication_thread

```
struct DDS_ThreadSettings_t DDS_AsynchronousPublisherQosPolicy::topic_query_publication_thread
```

<<***extension***>> *(p. 213)* Settings of the **DDSTopicQuery** (p. 1895) publication thread.

There is only one TopicQuery publication thread per **DDSPublisher** (p. 1818). This thread will exist as long as one or more **DDSDataWriter** (p. 1589) enables TopicQueries (via **DDS_DataWriterQos::topic_query_dispatch** (p. 917)).

[default] priority below normal.
The actual value depends on your architecture:

For Windows: -2
For Linux: OS default priority
For a complete list of platform specific values, please refer to `Platform Notes`. **[default]** The actual value depends on your architecture:

For Windows: OS default stack size
For Linux: OS default stack size
For a complete list of platform specific values, please refer to `Platform Notes`.

[default] mask = **DDS_THREAD_SETTINGS_KIND_MASK_DEFAULT** (p. 547)

Categories

Immutable (p. 333), **Concurrency** (p. 258), **Topic Queries** (p. 330)

9.14 DDS_AsyncWaitSetProperty_t Struct Reference

Specifies the **DDSAsyncWaitSet** (p. 1528) behavior.

Public Attributes

- struct **DDS_WaitSetProperty_t waitset_property**

  *Specifies the behavior of the asynchronous wait behavior, which is equivalent to the wait mechanism of a **DDSWaitSet** (p. 1897).*
- **DDS_UnsignedLong thread_pool_size**

  *Number of threads that comprise the thread pool of the **DDSAsyncWaitSet** (p. 1528).*
- struct **DDS_ThreadSettings_t thread_settings**

  ***DDS_ThreadSettings_t** (p. 1385) for each thread comprising the thread pool of the **DDSAsyncWaitSet** (p. 1528).*
- char * **thread_name_prefix**

  *Prefix used to compose the name of each thread that comprises the thread pool of the **DDSAsyncWaitSet** (p. 1528).*
- struct **DDS_Duration_t wait_timeout**

  Asynchronous wait timeout.
- **DDS_Long level**

  *Specifies the level of an **DDSAsyncWaitSet** (p. 1528).*

9.14.1 Detailed Description

Specifies the **DDSAsyncWaitSet** (p. 1528) behavior.

This property allows configuring the behavior of the asynchronous wait and the **DDSCondition** (p. 1544) dispatch, as well as the parameters of the thread pool.

See also

DDS_WaitSetProperty_t (p. 1509)

DDS_ThreadSettings_t (p. 1385)

9.14.2 Member Data Documentation

9.14.2.1 waitset_property

```
struct DDS_WaitSetProperty_t DDS_AsyncWaitSetProperty_t::waitset_property
```

Specifies the behavior of the asynchronous wait behavior, which is equivalent to the wait mechanism of a **DDSWaitSet** (p. 1897).

See also

DDS_WaitSetProperty_t (p. 1509)

9.14.2.2 thread_pool_size

```
DDS_UnsignedLong DDS_AsyncWaitSetProperty_t::thread_pool_size
```

Number of threads that comprise the thread pool of the **DDSAsyncWaitSet** (p. 1528).

Size must be equal to or greater than one.

[default] 1

9.14.2.3 thread_settings

```
struct DDS_ThreadSettings_t DDS_AsyncWaitSetProperty_t::thread_settings
```

DDS_ThreadSettings_t (p. 1385) for each thread comprising the thread pool of the **DDSAsyncWaitSet** (p. 1528).

Each thread within the pool is created with the same settings.

[default] Default thread settings values.

See also

DDS_ThreadSettings_t (p. 1385)

9.14.2.4 thread_name_prefix

```
char* DDS_AsyncWaitSetProperty_t::thread_name_prefix
```

Prefix used to compose the name of each thread that comprises the thread pool of the **DDSAsyncWaitSet** (p. 1528).

The composed name has the form:

`thread_name_prefix##[index]AWs` where [index] is an integer that identifies the thread relative to the **DDSAsyncWaitSet** (p. 1528).

If NULL, the default prefix will be used.

[default] NULL (use default prefix)

9.14.2.5 wait_timeout

```
struct DDS_Duration_t DDS_AsyncWaitSetProperty_t::wait_timeout
```

Asynchronous wait timeout.

Specifies the maximum amount of time the leader thread of the **DDSAsyncWaitSet** (p. 1528) waits for an attached **DDSCondition** (p. 1544) to trigger before it wakes up.

Duration must be a value greater than zero.

See also

DDSWaitSet::wait (p. 1901)

[default] DDS_DURATION_INFINITE (p. 522)

9.14.2.6 level

```
DDS_Long DDS_AsyncWaitSetProperty_t::level
```

Specifies the level of an **DDSAsyncWaitSet** (p. 1528).

The level prevents an application to deadlock when it uses multiple **DDSAsyncWaitSet** (p. 1528) instances that call operations on each other from the context of one of their thread pool's thread.

Inside the context of one of these threads, the application can synchronize only with other **DDSAsyncWaitSet** (p. 1528) of bigger level.

[default] 1

9.15 DDS_AvailabilityQosPolicy Struct Reference

<<***extension***>> *(p. 213)* Configures the availability of data.

Public Attributes

- **DDS_Boolean enable_required_subscriptions**

  <<***extension***>> *(p. 213) Enables support for required subscriptions in a **DDSDataWriter** (p. 1589)*
- struct **DDS_Duration_t max_data_availability_waiting_time**

  <<***extension***>> *(p. 213) Defines how much time to wait before delivering a sample to the application without having received some of the previous samples.*
- struct **DDS_Duration_t max_endpoint_availability_waiting_time**

  <<***extension***>> *(p. 213) Defines how much time to wait to discover DataWriters providing samples for the same data source (virtual GUID).*
- struct **DDS_EndpointGroupSeq required_matched_endpoint_groups**

  <<***extension***>> *(p. 213) A sequence of endpoint groups.*

9.15.1 Detailed Description

<<***extension***>> *(p. 213)* Configures the availability of data.

Entity:

DDSDataReader (p. 1556), **DDSDataWriter** (p. 1589)

Properties:

RxO (p. 556) = NO
Changeable (p. 556) = **YES** (p. 556) (only on a **DDSDataWriter** (p. 1589) except for the member **DDS_AvailabilityQosPolicy::enable_required_subscriptions** (p. 808))

9.15.2 Usage

This QoS policy is used in the context of two features:

- Collaborative DataWriters
- Required Subscriptions

Collaborative DataWriters

The Collaborative DataWriters feature allows having multiple DataWriters publishing samples from a common logical data source. The DataReaders will combine the samples coming from the DataWriters in order to reconstruct the correct order at the source.

This QoS policy allows you to configure the ordering and combination process in the DataReader and can be used to support two different use cases:

- **Ordered delivery of samples in high-availability scenarios** One example of this is RTI Persistence Service. When a late-joining DataReader configured with **DDS_DurabilityQosPolicy** (p. 1005) set to **DDS_PERSISTENT_DURABILITY_QOS** (p. 599) or **DDS_TRANSIENT_DURABILITY_QOS** (p. 599) joins a DDS domain, it will start receiving historical samples from multiple DataWriters. For example, if the original DataWriter is still alive, the newly created DataReader will receive samples from the original DataWriter and one or more RTI Persistence Service DataWriters (PRSTDataWriters). This policy can be used to configure the sample ordering process on the DataReader.

- **Ordered delivery of samples in load-balanced scenarios** Multiple instances of the same application can work together to process and deliver samples. When the samples arrive through different data-paths out of order, the DataReader will be able to reconstruct the order at the destination. An example of this is when multiple instances of RTI Persistence Service are used to persist the data. Persisting data to a database on disk can be a bottleneck for throughput. You can improve scalability and performance by dividing the workload across different instances of RTI Persistence Service that use different databases. For example, samples larger than 10 are persisted by Persistence Service 1, samples less than or equal to 10 are persisted by Persistence Service 2.

- **Ordered delivery of samples with Group Ordered Access** This policy can also be used to configure the sample ordering process when the Subscriber is configured with **DDS_PresentationQosPolicy** (p. 1249) access_scope set to **DDS_GROUP_PRESENTATION_QOS** (p. 616). In this case, the Subscriber must deliver in order the samples published by a group of DataWriters that belong to the same Publisher and have access_scope set to **DDS_GROUP_PRESENTATION_QOS** (p. 616).

Each sample published in a DDS domain for a given logical data source is uniquely identified by a pair (virtual GUID, virtual sequence number). Samples from the same data source (same virtual GUID) can be published by different DataWriters. A DataReader will deliver a sample (VGUIDn, VSNm) to the application if one of the following conditions is satisfied:

- (VGUIDn, VSNm-1) has already been delivered to the application.
- All the known DataWriters publishing VGUIDn have announced that they do not have (VGUIDn, VSNm-1).
- None of the known DataWriters publishing GUIDn have announced potential availability of (VGUIDn, VSNm-1) and both timeouts in this QoS policy have expired.

A DataWriter announces potential availability of samples by using virtual heartbeats (HBs).

When **DDS_PresentationQosPolicy::access_scope** (p. 1252) is set to **DDS_TOPIC_PRESENTATION_QOS** (p. 616) or **DDS_INSTANCE_PRESENTATION_QOS** (p. 616), the virtual HB contains information about the samples contained in the **DDSDataWriter** (p. 1589) history.

When **DDS_PresentationQosPolicy::access_scope** (p. 1252) is set to **DDS_GROUP_PRESENTATION_QOS** (p. 616), the virtual HB contains information about all DataWriters in the **DDSPublisher** (p. 1818).

The frequency at which virtual HBs are sent is controlled by the protocol parameters **DDS_RtpsReliableWriter↩ Protocol_t::virtual_heartbeat_period** (p. 1325) and **DDS_RtpsReliableWriterProtocol_t::samples_per_virtual_↩ heartbeat** (p. 1325).

Required Subscriptions

In the context of Required Subscriptions, this QoS policy can be used to configure a set of Required Subscriptions on a **DDSDataWriter** (p. 1589).

Required subscriptions are preconfigured, named subscriptions that may leave and subsequently rejoin the network from time to time, at the same or different physical locations. Any time a required subscription is disconnected, any samples that would have been delivered to it are stored for delivery if and when the subscription rejoins the network.

9.15.3 Consistency

For a DataWriter, the setting of **AVAILABILITY** (p. 561) must be set consistenly with that of the **RELIABILITY** (p. 631) and **DURABILITY** (p. 596).

If **DDS_AvailabilityQosPolicy::enable_required_subscriptions** (p. 808) is set to **DDS_BOOLEAN_TRUE** (p. 513), **DDS_ReliabilityQosPolicy::kind** (p. 1298) must be set to **DDS_RELIABLE_RELIABILITY_QOS** (p. 633), **DDS_↩ DurabilityQosPolicy** (p. 1005) must be set to a value different than **DDS_VOLATILE_DURABILITY_QOS** (p. 598), and **DDS_DurabilityQosPolicy::writer_depth** (p. 1008) must be set to either **DDS_AUTO_WRITER_DEPTH** (p. 599) or **DDS_LENGTH_UNLIMITED** (p. 636).

9.15.4 Member Data Documentation

9.15.4.1 enable_required_subscriptions

```
DDS_Boolean DDS_AvailabilityQosPolicy::enable_required_subscriptions
```

<<***extension***>> *(p. 213)* Enables support for required subscriptions in a **DDSDataWriter** (p. 1589)

[default] DDS_BOOLEAN_FALSE (p. 513)

Categories

Immutable (p. 333), **Reliability** (p. 314), **Durability** (p. 278), **Data Availability** (p. 261)

9.15.4.2 max_data_availability_waiting_time

```
struct DDS_Duration_t DDS_AvailabilityQosPolicy::max_data_availability_waiting_time
```

<<***extension***>> *(p. 213)* Defines how much time to wait before delivering a sample to the application without having received some of the previous samples.

Collaborative DataWriters

A sample identified by (VGUIDn, VSNm) will be delivered to the application if this timeout expires for the sample and the following two conditions are satisfied:

- None of the known DataWriters publishing VGUIDn have announced potential availability of (VGUIDn, VSNm-1).
- The DataWriters for all the endpoint groups specified in **required_matched_endpoint_groups** (p. 809) have been discovered or **max_endpoint_availability_waiting_time** (p. 808) has expired.

Required Subscriptions

This field is not applicable to Required Subscriptions.

[default] DDS_DURATION_AUTO (p. 523) (**DDS_DURATION_INFINITE** (p. 522) for **DDS_GROUP_PRESENTATION_QOS** (p. 616). Otherwise, 0 seconds)

[range] [0, **DDS_DURATION_INFINITE** (p. 522)], **DDS_DURATION_AUTO** (p. 523)

Categories

Mutable (p. 369), **Reliability** (p. 314), **Durability** (p. 278), **Data Availability** (p. 261)

9.15.4.3 max_endpoint_availability_waiting_time

`struct` **`DDS_Duration_t`** `DDS_AvailabilityQosPolicy::max_endpoint_availability_waiting_time`

<<***extension***>> *(p. 213)* Defines how much time to wait to discover DataWriters providing samples for the same data source (virtual GUID).

Collaborative DataWriters

The set of endpoint groups that are required to provide samples for a data source can be configured using **required_matched_endpoint_groups** (p. 809).

A non-consecutive sample identified by (VGUIDn, VSNm) cannot be delivered to the application unless DataWriters for all the endpoint groups in **required_matched_endpoint_groups** (p. 809) are discovered or this timeout expires.

Required Subscriptions

This field is not applicable to Required Subscriptions.

[default] DDS_DURATION_AUTO (p. 523) (**DDS_DURATION_INFINITE** (p. 522) for **DDS_GROUP_PRESENTATION_QOS** (p. 616). Otherwise, 0 seconds)

[range] [0, **DDS_DURATION_INFINITE** (p. 522)], **DDS_DURATION_AUTO** (p. 523)

Categories

Mutable (p. 369), **Reliability** (p. 314), **Durability** (p. 278), **Data Availability** (p. 261)

9.15.4.4 required_matched_endpoint_groups

`struct` **`DDS_EndpointGroupSeq`** `DDS_AvailabilityQosPolicy::required_matched_endpoint_groups`

<<***extension***>> *(p. 213)* A sequence of endpoint groups.

Collaborative DataWriters

In the context of Collaborative DataWriters, it specifies the set of endpoint groups that are expected to provide samples for the same data source.

The quorum count in a group represents the number of DataWriters that must be discovered for that group before the DataReader is allowed to provide non consecutive samples to the application.

A DataWriter becomes a member of an endpoint group by configuring the role_name in **DDS_DataWriterQos::publication_name** (p. 916).

Required Subscriptions

In the context of Required Subscriptions, it specifies the set of Required Subscriptions on a **DDSDataWriter** (p. 1589).

Each Required Subscription is specified by a name and a quorum count.

The quorum count represents the number of DataReaders that have to acknowledge the sample before it can be considered fully acknowledged for that Required Subscription.

A DataReader is associated with a Required Subscription by configuring the role_name in **DDS_DataReaderQos::subscription_name** (p. 865).

[default] Empty sequence

Categories

Mutable (p. 369), **Reliability** (p. 314), **Durability** (p. 278), **Data Availability** (p. 261)

9.16 DDS_BatchQosPolicy Struct Reference

*<<**extension**>> (p. 213)* Used to configure batching of multiple samples into a single network packet in order to increase throughput for small samples.

Public Attributes

- **DDS_Boolean enable**

  *<<**basic**>> (p. 201) <<**extension**>> (p. 213) Specifies whether or not batching is enabled.*
- **DDS_Long max_data_bytes**

  *<<**basic**>> (p. 201) <<**extension**>> (p. 213) The maximum cumulative length of all serialized samples in a batch.*
- **DDS_Long max_samples**

  *<<**basic**>> (p. 201) <<**extension**>> (p. 213) The maximum number of samples in a batch.*
- struct **DDS_Duration_t max_flush_delay**

  *<<**extension**>> (p. 213) The maximum flush delay.*
- struct **DDS_Duration_t source_timestamp_resolution**

  *<<**extension**>> (p. 213) Batch source timestamp resolution.*
- **DDS_Boolean thread_safe_write**

  *<<**extension**>> (p. 213) Determines whether or not the write operation is thread safe.*

9.16.1 Detailed Description

*<<**extension**>> (p. 213)* Used to configure batching of multiple samples into a single network packet in order to increase throughput for small samples.

This QoS policy configures the ability of the middleware to collect multiple user data samples to be sent in a single network packet, to take advantage of the efficiency of sending larger packets and thus increase effective throughput.

This QoS policy can be used to dramatically increase effective throughput for small data samples. Usually, throughput for small samples (size $<$ 2048 bytes) is limited by CPU capacity and not by network bandwidth. Batching many smaller samples to be sent in a single large packet will increase network utilization, and thus throughput, in terms of samples per second.

Entity:

DDSDataWriter (p. 1589)

Properties:

RxO (p. 556) = NO
Changeable (p. 556) = **UNTIL ENABLE** (p. 556)

9.16.2 Member Data Documentation

9.16.2.1 enable

```
DDS_Boolean DDS_BatchQosPolicy::enable
```

<<***basic***>> *(p. 201)* <<***extension***>> *(p. 213)* Specifies whether or not batching is enabled.

[default] DDS_BOOLEAN_FALSE (p. 513)

Categories

Immutable (p. 333), **Batching** (p. 257), **Throughput** (p. 327)

9.16.2.2 max_data_bytes

```
DDS_Long DDS_BatchQosPolicy::max_data_bytes
```

<<***basic***>> *(p. 201)* <<***extension***>> *(p. 213)* The maximum cumulative length of all serialized samples in a batch.

A batch is flushed automatically when this maximum is reached.

max_data_bytes does not include the meta data associated with the batch samples. Each sample has at least 8 bytes of meta data containing information such as the timestamp and sequence number. The meta data can be as large as 52 bytes for keyed topics and 20 bytes for unkeyed topics.

Note: Batches must contain whole samples. If a new batch is started and its initial sample causes the serialized size to exceed max_data_bytes, RTI Connext will send the sample in a single batch.

[default] 1024

[range] [1,**DDS_LENGTH_UNLIMITED** (p. 636)]

9.16.3 Consistency

The setting of **DDS_BatchQosPolicy::max_data_bytes** (p. 811) must be consistent with **DDS_BatchQosPolicy**↩ **::max_samples** (p. 811). For these two values to be consistent, they cannot be both **DDS_LENGTH_UNLIMITED** (p. 636).

Categories

Immutable (p. 333), **Batching** (p. 257), **Throughput** (p. 327)

9.16.3.1 max_samples

```
DDS_Long DDS_BatchQosPolicy::max_samples
```

<<***basic***>> *(p. 201)* <<***extension***>> *(p. 213)* The maximum number of samples in a batch.

A batch is flushed automatically when this maximum is reached.

[default] DDS_LENGTH_UNLIMITED (p. 636)

[range] [1,**DDS_LENGTH_UNLIMITED** (p. 636)]

9.16.4 Consistency

The setting of **DDS_BatchQosPolicy::max_samples** (p. 811) must be consistent with **DDS_BatchQosPolicy::max_data_bytes** (p. 811). For these two values to be consistent, they cannot be both **DDS_LENGTH_UNLIMITED** (p. 636).

Categories

Immutable (p. 333), **Batching** (p. 257), **Throughput** (p. 327)

9.16.4.1 max_flush_delay

```
struct DDS_Duration_t DDS_BatchQosPolicy::max_flush_delay
```

<<***extension***>> *(p. 213)* The maximum flush delay.

A batch is flushed automatically after the delay specified by this parameter.

The delay is measured from the time the first sample in the batch is written by the application.

[default] DDS_DURATION_INFINITE (p. 522)

[range] [0,**DDS_DURATION_INFINITE** (p. 522)]

9.16.5 Consistency

The setting of **DDS_BatchQosPolicy::max_flush_delay** (p. 812) must be consistent with **DDS_AsynchronousPublisherQosPolicy::disable_asynchronous_batch** (p. 800) and **DDS_BatchQosPolicy::thread_safe_write** (p. 813). If the delay is different than **DDS_DURATION_INFINITE** (p. 522), **DDS_AsynchronousPublisherQosPolicy::disable_asynchronous_batch** (p. 800) must be set to **DDS_BOOLEAN_FALSE** (p. 513) and **DDS_BatchQosPolicy::thread_safe_write** (p. 813) must be set to **DDS_BOOLEAN_TRUE** (p. 513).

Categories

Immutable (p. 333), **Batching** (p. 257), **Throughput** (p. 327)

9.16.5.1 source_timestamp_resolution

```
struct  DDS_Duration_t DDS_BatchQosPolicy::source_timestamp_resolution
```

<<***extension***>> *(p. 213)* Batch source timestamp resolution.

The value of this field determines how the source timestamp is associated with the samples in a batch.

A sample written with timestamp 't' inherits the source timestamp 't2' associated with the previous sample unless ('t' - 't2') > source_timestamp_resolution.

If source_timestamp_resolution is set to **DDS_DURATION_INFINITE** (p. 522), every sample in the batch will share the source timestamp associated with the first sample.

If source_timestamp_resolution is set to zero, every sample in the batch will contain its own source timestamp corresponding to the moment when the sample was written.

The performance of the batching process is better when source_timestamp_resolution is set to **DDS_DURATION_INFINITE** (p. 522).

[default] DDS_DURATION_INFINITE (p. 522)

[range] [0,**DDS_DURATION_INFINITE** (p. 522)]

9.16.6 Consistency

The setting of **DDS_BatchQosPolicy::source_timestamp_resolution** (p. 812) must be consistent with **DDS_BatchQosPolicy::thread_safe_write** (p. 813). If **DDS_BatchQosPolicy::thread_safe_write** (p. 813) is set to **DDS_BOOLEAN_FALSE** (p. 513), **DDS_BatchQosPolicy::source_timestamp_resolution** (p. 812) must be set to **DDS_DURATION_INFINITE** (p. 522).

Categories

Immutable (p. 333), **Batching** (p. 257), **Throughput** (p. 327), **Data Availability** (p. 261)

9.16.6.1 thread_safe_write

```
DDS_Boolean DDS_BatchQosPolicy::thread_safe_write
```

<<***extension***>> *(p. 213)* Determines whether or not the write operation is thread safe.

If this parameter is set to **DDS_BOOLEAN_TRUE** (p. 513), multiple threads can call write on the **DDSDataWriter** (p. 1589) concurrently.

[default] DDS_BOOLEAN_TRUE (p. 513)

9.16.7 Consistency

The setting of **DDS_BatchQosPolicy::thread_safe_write** (p. 813) must be consistent with **DDS_BatchQosPolicy::source_timestamp_resolution** (p. 812). If **DDS_BatchQosPolicy::thread_safe_write** (p. 813) is set to **DDS_BOOLEAN_FALSE** (p. 513), **DDS_BatchQosPolicy::source_timestamp_resolution** (p. 812) must be set to **DDS_DURATION_INFINITE** (p. 522).

Categories

Immutable (p. 333), **Batching** (p. 257), **Throughput** (p. 327), **Concurrency** (p. 258)

9.17 DDS_BooleanSeq Struct Reference

Instantiates **FooSeq** (p. 1965) < **DDS_Boolean** (p. 516) >

9.17.1 Detailed Description

Instantiates **FooSeq** (p. 1965) < **DDS_Boolean** (p. 516) >

Instantiates:

<<***generic***>> *(p. 433)* **FooSeq** (p. 1965)

See also

DDS_Boolean (p. 516)

FooSeq (p. 1965)

9.18 DDS_BuiltinTopicKey_t Struct Reference

The key type of the built-in topic types.

Public Attributes

- DDS_BUILTIN_TOPIC_KEY_TYPE_NATIVE **value** [DDS_BUILTIN_TOPIC_KEY_TYPE_NATIVE_LENGTH]

  *An array of four integers that uniquely represents a remote **DDSEntity** (p. 1731).*

9.18.1 Detailed Description

The key type of the built-in topic types.

Each remote **DDSEntity** (p. 1731) to be discovered can be uniquely identified by this key. This is the key of all the built-in topic data types.

See also

DDS_ParticipantBuiltinTopicData (p. 1229)

DDS_TopicBuiltinTopicData (p. 1391)

DDS_PublicationBuiltinTopicData (p. 1264)

DDS_SubscriptionBuiltinTopicData (p. 1370)

9.19 DDS_BuiltinTopicReaderResourceLimits_t Struct Reference

*<<**extension**>> (p. 213)* Built-in topic reader's resource limits.

Public Attributes

- **DDS_Long initial_samples**

  *<<**extension**>> (p. 213) Initial number of samples.*
- **DDS_Long max_samples**

  *<<**extension**>> (p. 213) Maximum number of samples.*
- **DDS_Long initial_infos**

  *<<**extension**>> (p. 213) Initial number of sample infos.*
- **DDS_Long max_infos**

  *<<**extension**>> (p. 213) Maximum number of sample infos.*
- **DDS_Long initial_outstanding_reads**

  *<<**extension**>> (p. 213) The initial number of outstanding reads that have not called finish yet on the same built-in topic **DDSDataReader** (p. 1556).*
- **DDS_Long max_outstanding_reads**

  *<<**extension**>> (p. 213) The maximum number of outstanding reads that have not called finish yet on the same built-in topic **DDSDataReader** (p. 1556).*
- **DDS_Long max_samples_per_read**

  *<<**extension**>> (p. 213) Maximum number of samples that can be read/taken on a same built-in topic **DDSDataReader** (p. 1556).*
- **DDS_Boolean disable_fragmentation_support**

  ***[DEPRECATED]** (p. **384**) <<**extension**>> (p. 213) Determines whether the built-in topic **DDSDataReader** (p. 1556) can receive fragmented samples.*
- **DDS_Long max_fragmented_samples**

  *<<**extension**>> (p. 213) The maximum number of samples for which the built-in topic **DDSDataReader** (p. 1556) may store fragments at a given point in time.*
- **DDS_Long initial_fragmented_samples**

  *<<**extension**>> (p. 213) The initial number of samples for which a built-in topic **DDSDataReader** (p. 1556) may store fragments.*

- **DDS_Long max_fragmented_samples_per_remote_writer**

  <<***extension***>> *(p. 213) The maximum number of samples per remote writer for which a built-in topic* ***DDSDataReader*** *(p. 1556) may store fragments.*
- **DDS_Long max_fragments_per_sample**

  <<***extension***>> *(p. 213) Maximum number of fragments for a single sample.*
- **DDS_Boolean dynamically_allocate_fragmented_samples**

  <<***extension***>> *(p. 213) Determines whether the built-in topic* ***DDSDataReader*** *(p. 1556) pre-allocates storage for storing fragmented samples.*

9.19.1 Detailed Description

<<***extension***>> *(p. 213)* Built-in topic reader's resource limits.

Defines the resources that can be used for a built-in-topic data reader.

A built-in topic data reader subscribes reliably to built-in topics containing declarations of new entities or updates to existing entities in the domain. Keys are used to differentiate among entities of the same type. RTI Connext assigns a unique key to each entity in a domain.

Properties:

RxO (p. 556) = N/A
Changeable (p. 556) = **NO** (p. 556)

QoS:

DDS_DiscoveryConfigQosPolicy (p. 934)

9.19.2 Member Data Documentation

9.19.2.1 initial_samples

```
DDS_Long DDS_BuiltinTopicReaderResourceLimits_t::initial_samples
```

<<***extension***>> *(p. 213)* Initial number of samples.

This should be a value between 1 and initial number of instance of the built-in-topic reader, depending on how many instances are sending data concurrently.

[default] 64

[range] [1, 1 million], <= max_samples

Categories

Immutable (p. 333), **Sample Memory Management** (p. 321), **Middleware Memory Management** (p. 297), **Durability** (p. 278)

9.19.2.2 max_samples

```
DDS_Long DDS_BuiltinTopicReaderResourceLimits_t::max_samples
```

<<***extension***>> *(p. 213)* Maximum number of samples.

This should be a value between 1 and max number of instance of the built-in-topic reader, depending on how many instances are sending data concurrently. Also, it should not be less than initial_samples.

[default] DDS_LENGTH_UNLIMITED (p. 636)

[range] [1, 1 million] or **DDS_LENGTH_UNLIMITED** (p. 636), >= initial_samples

Categories

Immutable (p. 333), **Sample Memory Management** (p. 321), **Middleware Memory Management** (p. 297), **Durability** (p. 278)

9.19.2.3 initial_infos

```
DDS_Long DDS_BuiltinTopicReaderResourceLimits_t::initial_infos
```

<<***extension***>> *(p. 213)* Initial number of sample infos.

The initial number of info units that a built-in topic **DDSDataReader** (p. 1556) can have. Info units are used to store **DDS_SampleInfo** (p. 1344).

[default] 64

[range] [1, 1 million] <= max_infos

Categories

Immutable (p. 333), **Middleware Memory Management** (p. 297)

9.19.2.4 max_infos

```
DDS_Long DDS_BuiltinTopicReaderResourceLimits_t::max_infos
```

<<***extension***>> *(p. 213)* Maximum number of sample infos.

The maximum number of info units that a built-in topic **DDSDataReader** (p. 1556) can use to store **DDS_SampleInfo** (p. 1344).

[default] DDS_LENGTH_UNLIMITED (p. 636)

[range] [1, 1 million] or **DDS_LENGTH_UNLIMITED** (p. 636), >= initial_infos

Categories

Immutable (p. 333), **Middleware Memory Management** (p. 297)

9.19.2.5 initial_outstanding_reads

```
DDS_Long DDS_BuiltinTopicReaderResourceLimits_t::initial_outstanding_reads
```

<<***extension***>> *(p. 213)* The initial number of outstanding reads that have not called finish yet on the same built-in topic **DDSDataReader** (p. 1556).

Must be less than or equal to max_outstanding_reads.

[default] 2

[range] [1, 1024]

Categories

Immutable (p. 333), **Middleware Memory Management** (p. 297), **Data Availability** (p. 261)

9.19.2.6 max_outstanding_reads

```
DDS_Long DDS_BuiltinTopicReaderResourceLimits_t::max_outstanding_reads
```

<<***extension***>> *(p. 213)* The maximum number of outstanding reads that have not called finish yet on the same built-in topic **DDSDataReader** (p. 1556).

Must be greater than or equal to initial_outstanding_reads.

[default] DDS_LENGTH_UNLIMITED (p. 636)

[range] [1, 1024] or **DDS_LENGTH_UNLIMITED** (p. 636)

Categories

Immutable (p. 333), **Middleware Memory Management** (p. 297), **Data Availability** (p. 261)

9.19.2.7 max_samples_per_read

```
DDS_Long DDS_BuiltinTopicReaderResourceLimits_t::max_samples_per_read
```

<<***extension***>> *(p. 213)* Maximum number of samples that can be read/taken on a same built-in topic **DDSData↩Reader** (p. 1556).

[default] 1024

[range] [1, 65536]

Categories

Immutable (p. 333), **Middleware Memory Management** (p. 297), **Data Availability** (p. 261)

9.19.2.8 disable_fragmentation_support

```
DDS_Boolean DDS_BuiltinTopicReaderResourceLimits_t::disable_fragmentation_support
```

[DEPRECATED] (p. **384**) <<***extension***>> *(p. 213)* Determines whether the built-in topic **DDSDataReader** (p. 1556) can receive fragmented samples.

When fragmentation support is not needed, disabling fragmentation support will save some memory resources.

[default] DDS_BOOLEAN_FALSE (p. 513)

Categories

Immutable (p. 333), **Middleware Memory Management** (p. 297), **Fragmentation** (p. 284), **Sending Large Data** (p. 290)

9.19.2.9 max_fragmented_samples

```
DDS_Long DDS_BuiltinTopicReaderResourceLimits_t::max_fragmented_samples
```

<<***extension***>> *(p. 213)* The maximum number of samples for which the built-in topic **DDSDataReader** (p. 1556) may store fragments at a given point in time.

At any given time, a built-in topic **DDSDataReader** (p. 1556) may store fragments for up to `max_fragmented_samples` samples while waiting for the remaining fragments. These samples need not have consecutive sequence numbers and may have been sent by different built-in topic **DDSDataWriter** (p. 1589) instances.

Once all fragments of a sample have been received, the sample is treated as a regular sample and becomes subject to standard QoS settings such as **DDS_BuiltinTopicReaderResourceLimits_t::max_samples** (p. 816).

The middleware will drop fragments if the `max_fragmented_samples` limit has been reached. For best-effort communication, the middleware will accept a fragment for a new sample, but drop the oldest fragmented sample from the same remote writer. For reliable communication, the middleware will drop fragments for any new samples until all fragments for at least one older sample from that writer have been received.

Only applies if **DDS_BuiltinTopicReaderResourceLimits_t::disable_fragmentation_support** (p. 818) is **DDS_BOOLEAN_FALSE** (p. 513).

[default] 1024

[range] [1, 1 million]

Categories

Immutable (p. 333), **Middleware Memory Management** (p. 297), **Fragmentation** (p. 284), **Sending Large Data** (p. 290)

9.19.2.10 initial_fragmented_samples

```
DDS_Long DDS_BuiltinTopicReaderResourceLimits_t::initial_fragmented_samples
```

<<***extension***>> *(p. 213)* The initial number of samples for which a built-in topic **DDSDataReader** (p. 1556) may store fragments.

Only applies if **DDS_BuiltinTopicReaderResourceLimits_t::disable_fragmentation_support** (p. 818) is **DDS_BOOLEAN_FALSE** (p. 513).

[default] 4

[range] [1,1024], <= max_fragmented_samples

Categories

Immutable (p. 333), **Middleware Memory Management** (p. 297), **Fragmentation** (p. 284), **Sending Large Data** (p. 290)

9.19.2.11 max_fragmented_samples_per_remote_writer

```
DDS_Long DDS_BuiltinTopicReaderResourceLimits_t::max_fragmented_samples_per_remote_writer
```

<<***extension***>> *(p. 213)* The maximum number of samples per remote writer for which a built-in topic **DDSDataReader** (p. 1556) may store fragments.

Logical limit so a single remote writer cannot consume all available resources.

Only applies if **DDS_BuiltinTopicReaderResourceLimits_t::disable_fragmentation_support** (p. 818) is **DDS_BOOLEAN_FALSE** (p. 513).

[default] 256

[range] [1, 1 million], <= max_fragmented_samples

Categories

Immutable (p. 333), **Middleware Memory Management** (p. 297), **Fragmentation** (p. 284), **Sending Large Data** (p. 290)

9.19.2.12 max_fragments_per_sample

```
DDS_Long DDS_BuiltinTopicReaderResourceLimits_t::max_fragments_per_sample
```

<<***extension***>> *(p. 213)* Maximum number of fragments for a single sample.

Only applies if **DDS_BuiltinTopicReaderResourceLimits_t::disable_fragmentation_support** (p. 818) is **DDS_↩ BOOLEAN_FALSE** (p. 513).

[default] DDS_LENGTH_UNLIMITED (p. 636)

[range] [1, 1 million] or **DDS_LENGTH_UNLIMITED** (p. 636)

Categories

Immutable (p. 333), **Middleware Memory Management** (p. 297), **Fragmentation** (p. 284), **Sending Large Data** (p. 290)

9.19.2.13 dynamically_allocate_fragmented_samples

```
DDS_Boolean DDS_BuiltinTopicReaderResourceLimits_t::dynamically_allocate_fragmented_samples
```

<<***extension***>> *(p. 213)* Determines whether the built-in topic **DDSDataReader** (p. 1556) pre-allocates storage for storing fragmented samples.

By default, the middleware does not allocate memory upfront, but instead allocates memory from the heap upon receiving the first fragment of a new sample. The amount of memory allocated equals the amount of memory needed to store all fragments in the sample. Once all fragments of a sample have been received, the sample is deserialized and stored in the regular receive queue. At that time, the dynamically allocated memory is freed again.

This QoS setting is useful for large, but variable-sized data types where upfront memory allocation for multiple samples based on the maximum possible sample size may be expensive. The main disadvantage of not pre-allocating memory is that one can no longer guarantee the middleware will have sufficient resources at runtime.

If `dynamically_allocate_fragmented_samples` is set to **DDS_BOOLEAN_FALSE** (p. 513), the middleware will allocate memory upfront for storing fragments for up to **DDS_DataReaderResourceLimitsQosPolicy↩ ::initial_fragmented_samples** (p. 875) samples. This memory may grow up to **DDS_DataReaderResourceLimits↩ QosPolicy::max_fragmented_samples** (p. 874) if needed.

Only applies if **DDS_DataReaderResourceLimitsQosPolicy::disable_fragmentation_support** (p. 874) is **DDS_↩ BOOLEAN_FALSE** (p. 513).

[default] DDS_BOOLEAN_TRUE (p. 513)

Categories

Immutable (p. 333), **Middleware Memory Management** (p. 297), **Fragmentation** (p. 284), **Sending Large Data** (p. 290)

9.20 DDS_ChannelSettings_t Struct Reference

<<***extension***>> *(p. 213)* Type used to configure the properties of a channel.

Public Attributes

- struct **DDS_TransportMulticastSettingsSeq multicast_settings**

  <<***extension***>> *(p. 213) A sequence of* ***DDS_TransportMulticastSettings_t*** *(p. 1417) used to configure the multicast addresses associated with a channel.*
- char * **filter_expression**

  <<***extension***>> *(p. 213) A logical expression used to determine the data that will be published in the channel.*
- **DDS_Long priority**

  <<***extension***>> *(p. 213) Publication priority*

9.20.1 Detailed Description

<<***extension***>> *(p. 213)* Type used to configure the properties of a channel.

QoS:

DDS_MultiChannelQosPolicy (p. 1214)

9.20.2 Member Data Documentation

9.20.2.1 multicast_settings

```
struct DDS_TransportMulticastSettingsSeq DDS_ChannelSettings_t::multicast_settings
```

<<***extension***>> *(p. 213)* A sequence of **DDS_TransportMulticastSettings_t** (p. 1417) used to configure the multicast addresses associated with a channel.

The sequence cannot be empty.

The maximum number of multicast locators in a channel is limited to 16 (a locator is defined by a transport alias, a multicast address and a port). Note that this is a hard limit that cannot be increased. However, this limit can be decreased by configuring the 'dds.domain_participant.max_announced_locator_list_size' property in the **DDS_PropertyQosPolicy** (p. 1260) associated with the **DDS_DomainParticipantQos** (p. 971).

[default] Empty sequence (invalid value)

Categories

Immutable (p. 333), **Transports** (p. 327), **Content Filtering** (p. 260)

9.20.2.2 filter_expression

```
char* DDS_ChannelSettings_t::filter_expression
```

<<***extension***>> *(p. 213)* A logical expression used to determine the data that will be published in the channel.

If the expression evaluates to TRUE, a sample will be published on the channel.

An empty string always evaluates the expression to TRUE.

A NULL value is not allowed.

The syntax of the expression will depend on the value of **DDS_MultiChannelQosPolicy::filter_name** (p. 1215)

Important: This value must be an allocated string with **DDS_String_alloc** (p. 759) or **DDS_String_dup** (p. 761). It should not be assigned to a string constant.

The filter expression length (including NULL-terminated character) cannot be greater than **DDS_DomainParticipant↩ ResourceLimitsQosPolicy::channel_filter_expression_max_length** (p. 998).

See also

Queries and Filters Syntax (p. 184)

[default] NULL (invalid value)

Categories

Immutable (p. 333), **Content Filtering** (p. 260)

9.20.2.3 priority

```
DDS_Long DDS_ChannelSettings_t::priority
```

<<***extension***>> *(p. 213)* Publication priority

A positive integer value designating the relative priority of the channel, used to determine the transmission order of pending writes.

Use of publication priorities requires the asynchronous publisher (**DDS_ASYNCHRONOUS_PUBLISH_MODE↩ _QOS** (p. 629)) with **DDS_FlowControllerProperty_t::scheduling_policy** (p. 1157) set to **DDS_HPF_FLOW_↩ CONTROLLER_SCHED_POLICY** (p. 125).

Larger numbers have higher priority.

If the publication priority of the channel is any value other than **DDS_PUBLICATION_PRIORITY_UNDEFINED** (p. 628), then the channel's priority will take precedence over the data writer's priority.

If the publication priority of the channel is set to **DDS_PUBLICATION_PRIORITY_UNDEFINED** (p. 628), then the channel's priority will be set to the value of the data writer's priority.

If the publicaton priority of both the data writer and the channel are **DDS_PUBLICATION_PRIORITY_UNDEFINED** (p. 628), the channel will be assigned the lowest priority value.

If the publication priority of the channel is **DDS_PUBLICATION_PRIORITY_AUTOMATIC** (p. 628), then the channel will be assigned the priority of the largest publication priority of all samples in the channel. The publication priority of each sample can be set in the **DDS_WriteParams_t** (p. 1519) of the **FooDataWriter::write_w_params** (p. 1956) function.

[default] DDS_PUBLICATION_PRIORITY_UNDEFINED (p. 628)

Categories

Immutable (p. 333), **Transports** (p. 327), **Data Availability** (p. 261)

9.21 DDS_ChannelSettingsSeq Struct Reference

Declares IDL sequence< **DDS_ChannelSettings_t** (p. 822) >

9.21.1 Detailed Description

Declares IDL sequence< **DDS_ChannelSettings_t** (p. 822) >

A sequence of **DDS_ChannelSettings_t** (p. 822) used to configure the channels' properties. If the length of the sequence is zero, the **DDS_MultiChannelQosPolicy** (p. 1214) has no effect.

Instantiates:

<<***generic***>> *(*p. *433)* **FooSeq** (p. 1965)

See also

DDS_ChannelSettings_t (p. 822)

9.22 DDS_CharSeq Struct Reference

Instantiates **FooSeq** (p. 1965) < **DDS_Char** (p. 513) >

9.22.1 Detailed Description

Instantiates **FooSeq** (p. 1965) < **DDS_Char** (p. 513) >

Instantiates:

<<***generic***>> *(*p. *433)* **FooSeq** (p. 1965)

See also

DDS_Char (p. 513)

FooSeq (p. 1965)

9.23 DDS_CoherentSetInfo_t Struct Reference

<<***extension***>> *(*p. *432)* Type definition for a coherent set info.

Public Attributes

- **DDS_GUID_t group_guid**

  The coherent set group GUID.
- **DDS_SequenceNumber_t coherent_set_sequence_number**

  *Sequence number that identifies a sample as part of a **DDSDataWriter** (p. 1589) coherent set.*
- **DDS_SequenceNumber_t group_coherent_set_sequence_number**

  Sequence number that identifies a sample as part of a group coherent set.
- **DDS_Boolean incomplete_coherent_set**

  Indicates if a sample is part of an incomplete coherent set.

9.23.1 Detailed Description

<<***extension***>> *(p. 432)* Type definition for a coherent set info.

A CoherentSetInfo provides information about the coherent set associated with a sample.

9.23.2 Member Data Documentation

9.23.2.1 group_guid

DDS_GUID_t DDS_CoherentSetInfo_t::group_guid

The coherent set group GUID.

This GUID identifies the **DDSDataWriter** (p. 1589) or the group of DataWriters publishing the coherent set, depending on the value of **DDS_PresentationQosPolicy::access_scope** (p. 1252) in the **DDSSubscriber** (p. 1860).

9.23.2.2 coherent_set_sequence_number

DDS_SequenceNumber_t DDS_CoherentSetInfo_t::coherent_set_sequence_number

Sequence number that identifies a sample as part of a **DDSDataWriter** (p. 1589) coherent set.

When **DDS_PresentationQosPolicy::access_scope** (p. 1252) in the **DDSSubscriber** (p. 1860) is set to **DDS_↩ INSTANCE_PRESENTATION_QOS** (p. 616) or **DDS_TOPIC_PRESENTATION_QOS** (p. 616), the coherent set associated with a sample is identified by the pair (group_guid, coherent_set_sequence_number).

9.23.2.3 group_coherent_set_sequence_number

```
DDS_SequenceNumber_t DDS_CoherentSetInfo_t::group_coherent_set_sequence_number
```

Sequence number that identifies a sample as part of a group coherent set.

When **DDS_PresentationQosPolicy::access_scope** (p. 1252) in the **DDSSubscriber** (p. 1860) is set to **DDS_↩ GROUP_PRESENTATION_QOS** (p. 616), the coherent set associated with a sample is identified by the pair (group_↩ guid, group_coherent_set_sequence_number).

9.23.2.4 incomplete_coherent_set

```
DDS_Boolean DDS_CoherentSetInfo_t::incomplete_coherent_set
```

Indicates if a sample is part of an incomplete coherent set.

9.24 DDS_CompressionSettings_t Struct Reference

*<<**extension**>> (p. 213)* Settings related to compressing user data.

Public Attributes

- **DDS_CompressionIdMask compression_ids**
  *<<**extension**>> (p. 213) Mask that represents the compression algorithms enabled.*
- **DDS_UnsignedLong writer_compression_level**
  *<<**extension**>> (p. 213) The level of compression to use when compressing data.*
- **DDS_Long writer_compression_threshold**
  *<<**extension**>> (p. 213) The threshold, in bytes, above which a serialized sample will be eligible to be compressed.*

9.24.1 Detailed Description

*<<**extension**>> (p. 213)* Settings related to compressing user data.

QoS:

DDS_DataRepresentationQosPolicy (p. 886)

9.24.2 Member Data Documentation

9.24.2.1 compression_ids

DDS_CompressionIdMask DDS_CompressionSettings_t::compression_ids

<<***extension***>> *(p. 213)* Mask that represents the compression algorithms enabled.

A bitmap that represents the compression algorithm IDs (**DDS_CompressionIdMask** (p. 573)) that are supported by the endpoint. The **DDSDataWriter** (p. 1589) creation will fail if more than one algorithm is provided.

If a **DDSDataWriter** (p. 1589) inherits multiple compression IDs from a **DDSTopic** (p. 1885), only the least significant bit enabled will be inherited. This forces the following order of preference: **DDS_COMPRESSION_ID_ZLIB** (p. 573), **DDS_COMPRESSION_ID_BZIP2** (p. 573), **DDS_COMPRESSION_ID_LZ4** (p. 574).

Interactions with Security and Batching: Currently, the only algorithm that is supported when compression and batching are enabled on the same **DDSDataWriter** (p. 1589) is **DDS_COMPRESSION_ID_ZLIB** (p. 573).

The combination of compression, batching, and data protection is supported. First, compression is applied to the entire batch. Then, data protection is applied to the compressed batch.

Note: When **DDS_DataWriterProtocolQosPolicy::serialize_key_with_dispose** (p. 895) is enabled and a dispose message is sent, the serialized key is not compressed.

[default] For **DDSTopic** (p. 1885), **DDSDataWriter** (p. 1589) a **DDS_CompressionIdMask** (p. 573) mask set to **DDS_COMPRESSION_ID_MASK_NONE** (p. 571)

[default] For **DDSDataReader** (p. 1556) a **DDS_CompressionIdMask** (p. 573) mask set to **DDS_COMPRESSION_ID_MASK_ALL** (p. 571).

Categories

Immutable (p. 333), **Matching** (p. 382), **Discovery Metadata** (p. 377), **Sending Large Data** (p. 290)

9.24.2.2 writer_compression_level

DDS_UnsignedLong DDS_CompressionSettings_t::writer_compression_level

<<***extension***>> *(p. 213)* The level of compression to use when compressing data.

Compression algorithms typically allow you to choose a level with which to compress the data. Each level has trade-offs between the resulting compression ratio and the speed of compression.

[range] [0, 10]

The value 1 represents the fastest compression time and the lowest compression ratio. The value 10 represents the slowest compression time but the highest compression ratio.

A value of 0 disables compression.

[default] DDS_COMPRESSION_LEVEL_BEST_COMPRESSION (p. 571)

Note

Only available for a **DDSDataWriter** (p. 1589) and **DDSTopic** (p. 1885).

Categories

Immutable (p. 333), **Sending Large Data** (p. 290)

9.24.2.3 writer_compression_threshold

```
DDS_Long DDS_CompressionSettings_t::writer_compression_threshold
```

<<***extension***>> *(p. 213)* The threshold, in bytes, above which a serialized sample will be eligible to be compressed.

Any sample with a serialized size greater than or equal to the threshold will be eligible to be compressed. All samples with an eligible serialized size will be compressed. Only if the compressed size is smaller than the serialized size will the sample be stored and sent compressed on the wire.

For batching we check the maximum serialized size of the batch, calculated as serialized_sample_max_size * **DDS_BatchQosPolicy::max_samples** (p. 811)

[range] [0, 2147483647] or **DDS_LENGTH_UNLIMITED** (p. 636)

Setting the threshold to **DDS_LENGTH_UNLIMITED** (p. 636) disables the compression.

[default] DDS_COMPRESSION_THRESHOLD_DEFAULT (p. 572) (8192)

Note

Only available for a **DDSDataWriter** (p. 1589) and **DDSTopic** (p. 1885).

Categories

Immutable (p. 333), **Sending Large Data** (p. 290)

9.25 DDS_ContentFilterProperty_t Struct Reference

<<***extension***>> *(p. 432)* Type used to provide all the required information to enable content filtering.

Public Attributes

- char * **content_filter_topic_name**

  Name of the Content-filtered Topic associated with the Reader.
- char * **related_topic_name**

  Name of the Topic related to the Content-filtered Topic.
- char * **filter_class_name**

  Identifies the filter class this filter belongs to. RTPS can support multiple filter classes (SQL, regular expressions, custom filters, etc).
- char * **filter_expression**

  The actual filter expression. Must be a valid expression for the filter class specified using filterClassName.
- struct **DDS_StringSeq expression_parameters**

  Defines the value for each parameter in the filter expression.

9.25.1 Detailed Description

<<***extension***>> *(p. 432)* Type used to provide all the required information to enable content filtering.

9.25.2 Member Data Documentation

9.25.2.1 content_filter_topic_name

```
char* DDS_ContentFilterProperty_t::content_filter_topic_name
```

Name of the Content-filtered Topic associated with the Reader.

9.25.2.2 related_topic_name

```
char* DDS_ContentFilterProperty_t::related_topic_name
```

Name of the Topic related to the Content-filtered Topic.

9.25.2.3 filter_class_name

```
char* DDS_ContentFilterProperty_t::filter_class_name
```

Identifies the filter class this filter belongs to. RTPS can support multiple filter classes (SQL, regular expressions, custom filters, etc).

9.25.2.4 filter_expression

```
char* DDS_ContentFilterProperty_t::filter_expression
```

The actual filter expression. Must be a valid expression for the filter class specified using filterClassName.

9.25.2.5 expression_parameters

```
struct DDS_StringSeq DDS_ContentFilterProperty_t::expression_parameters
```

Defines the value for each parameter in the filter expression.

9.26 DDS_Cookie_t Struct Reference

<<***extension***>> *(*p. *432)* Sequence of bytes.

Public Member Functions

- void * **to_pointer** () const
  Returns the pointer previously stored in the cookie.

Public Attributes

- struct **DDS_OctetSeq value**
  a sequence of octets

9.26.1 Detailed Description

<<***extension***>> *(*p. *432)* Sequence of bytes.

9.26.2 Member Data Documentation

9.26.2.1 value

```
struct DDS_OctetSeq DDS_Cookie_t::value
```

a sequence of octets

[default] Empty (zero-sized)

9.27 DDS_CookieSeq Struct Reference

Declares IDL `sequence` < **DDS_Cookie_t** (p. 830) > .

9.27.1 Detailed Description

Declares IDL `sequence` < **DDS_Cookie_t** (p. 830) > .

See also

FooSeq (p. 1965)

9.28 DDS_DatabaseQosPolicy Struct Reference

<<***extension***>> *(p. 213)* Various threads and resource limits settings used by RTI Connext to control its internal database.

Public Attributes

- struct **DDS_ThreadSettings_t thread**

  <<***extension***>> *(p. 213) Database thread settings.*
- struct **DDS_Duration_t shutdown_timeout**

  [DEPRECATED] *(p. **384**)* <<***extension***>> *(p. 213) The maximum wait time during a shutdown.*
- struct **DDS_Duration_t cleanup_period**

  [DEPRECATED] *(p. **384**)* <<***extension***>> *(p. 213) The period at which the database thread wakes up to remove deleted records.*
- struct **DDS_Duration_t shutdown_cleanup_period**

  [DEPRECATED] *(p. **384**)* <<***extension***>> *(p. 213) The clean-up period used during database shut-down.*
- **DDS_Long initial_records**

  <<***extension***>> *(p. 213) The initial number of total records.*
- **DDS_Long max_skiplist_level**

  [DEPRECATED] *(p. **384**)* <<***extension***>> *(p. 213) The maximum level of the skiplist.*
- **DDS_Long max_weak_references**

  <<***extension***>> *(p. 213) The maximum number of weak references.*
- **DDS_Long initial_weak_references**

  <<***extension***>> *(p. 213) The initial number of weak references.*

9.28.1 Detailed Description

<<***extension***>> *(p. 213)* Various threads and resource limits settings used by RTI Connext to control its internal database.

RTI uses an internal in-memory "database" to store information about entities created locally as well as remote entities found during the discovery process. This database uses a background thread to garbage-collect records related to deleted entities. When the **DDSDomainParticipant** (p. 1619) that maintains this database is deleted, it shuts down this thread.

The Database QoS policy is used to configure how RTI Connext manages its database, including how often it cleans up, the priority of the database thread, and limits on resources that may be allocated by the database.

You may be interested in modifying the **DDS_DatabaseQosPolicy::shutdown_timeout** (p. 832) and **DDS_Database↩ QosPolicy::shutdown_cleanup_period** (p. 833) parameters to decrease the time it takes to delete a **DDSDomain↩ Participant** (p. 1619) when your application is shutting down.

The **DDS_DomainParticipantResourceLimitsQosPolicy** (p. 977) controls the memory allocation for elements stored in the database.

This QoS policy is an extension to the DDS standard.

Entity:

DDSDomainParticipant (p. 1619)

Properties:

RxO (p. 556) = N/A
Changeable (p. 556) **NO** (p. 556)

9.28.2 Member Data Documentation

9.28.2.1 thread

`struct` **`DDS_ThreadSettings_t`** `DDS_DatabaseQosPolicy::thread`

<<***extension***>> *(p. 213)* Database thread settings.

There is only one database thread: the clean-up thread.

[default] Priority: LOW.
The actual value depends on your architecture:

For Windows: -3
For Linux: OS default priority
For a complete list of platform specific values, please refer to `Platform Notes`.

[default] Stack Size: The actual value depends on your architecture:

For Windows: OS default stack size
For Linux: OS default stack size
For a complete list of platform specific values, please refer to `Platform Notes`.

[default] Mask: **DDS_THREAD_SETTINGS_STDIO** (p. 548)

Categories

Immutable (p. 333), **Concurrency** (p. 258)

9.28.2.2 shutdown_timeout

`struct` **`DDS_Duration_t`** `DDS_DatabaseQosPolicy::shutdown_timeout`

[DEPRECATED] (p. **384**) <<***extension***>> *(p. 213)* The maximum wait time during a shutdown.

The domain participant will exit after the timeout, even if the database has not been fully cleaned up.

[default] 15 seconds

[range] [0,**DDS_DURATION_INFINITE** (p. 522)]

Categories

Immutable (p. 333)

9.28.2.3 cleanup_period

`struct` **`DDS_Duration_t`** `DDS_DatabaseQosPolicy::cleanup_period`

[DEPRECATED] (p. **384**) <<***extension***>> *(p. 213)* The period at which the database thread wakes up to remove deleted records.

[default] 61 seconds

[range] [0,1 year]

Categories

Immutable (p. 333)

9.28.2.4 shutdown_cleanup_period

`struct` **`DDS_Duration_t`** `DDS_DatabaseQosPolicy::shutdown_cleanup_period`

[DEPRECATED] (p. **384**) <<***extension***>> *(p. 213)* The clean-up period used during database shut-down.

If you would like to shorten the time taken for a DomainParticipant to shutdown, you can decrease this value.

It is recommended to set this value to something other than 0 if running in an RTOS environment, to avoid CPU starvation.

[default] 10 milliseconds

[range] [0,1 year]

Categories

Immutable (p. 333)

9.28.2.5 initial_records

```
DDS_Long DDS_DatabaseQosPolicy::initial_records
```

<<***extension***>> *(p. 213)* The initial number of total records.

[default] 1024

[range] [1,10 million]

Categories

Immutable (p. 333), **Middleware Memory Management** (p. 297)

9.28.2.6 max_skiplist_level

```
DDS_Long DDS_DatabaseQosPolicy::max_skiplist_level
```

[DEPRECATED] (p. 384) <<***extension***>> *(p. 213)* The maximum level of the skiplist.

The skiplist is used to keep records in the database. Usually, the search time is log2(N), where N is the total number of records in one skiplist. However, once N exceeds 2^n, where n is the maximum skiplist level, the search time will become more and more linear. Therefore, the maximum level should be set such that 2^n is larger than the maximum(N among all skiplists). Usually, the maximum N is the maximum number of remote and local writers or readers.

[default] 7

[range] [1,31]

Categories

Immutable (p. 333), **Middleware Memory Management** (p. 297)

9.28.2.7 max_weak_references

```
DDS_Long DDS_DatabaseQosPolicy::max_weak_references
```

<<***extension***>> *(p. 213)* The maximum number of weak references.

A weak reference is an internal data structure that refers to a record within RTI Connext' internal database. This field configures the maximum number of such references that RTI Connext may create.

The actual number of weak references is permitted to grow from an initial value (indicated by **DDS_DatabaseQos↩ Policy::initial_weak_references** (p. 835)) to this maximum. To prevent RTI Connext from allocating any weak references after the system has reached a steady state, set the initial and maximum values equal to one another. To indicate that the number of weak references should continue to grow as needed indefinitely, set this field to **DDS_LENGTH↩ _UNLIMITED** (p. 636). Be aware that although a single weak reference occupies very little memory, allocating a very large number of them can have a significant impact on your overall memory usage.

Tuning this value precisely is difficult without intimate knowledge of the structure of RTI Connext' database; doing so is an advanced feature not required by most applications. The default value has been chosen to be sufficient for reasonably large systems. If you believe you may need to modify this value, please consult with RTI support personnel for assistance.

[default] DDS_LENGTH_UNLIMITED (p. 636)

[range] [1, 100 million] or **DDS_LENGTH_UNLIMITED** (p. 636), >= initial_weak_references

See also

DDS_DatabaseQosPolicy::initial_weak_references (p. 835)

Categories

Immutable (p. 333), **Middleware Memory Management** (p. 297)

9.28.2.8 initial_weak_references

```
DDS_Long DDS_DatabaseQosPolicy::initial_weak_references
```

<<***extension***>> *(p. 213)* The initial number of weak references.

See **DDS_DatabaseQosPolicy::max_weak_references** (p. 834) for more information about what a weak reference is.

If the QoS set contains an initial_weak_references value that is too small to ever grow to **DDS_DatabaseQosPolicy↩ ::max_weak_references** (p. 834) using RTI Connext' internal algorithm, this value will be adjusted upwards as necessary. Subsequent accesses of this value will reveal the actual initial value used.

Changing the value of this field is an advanced feature; it is recommended that you consult with RTI support personnel before doing so.

[default] 2049, which is the minimum initial value imposed by RTI Connext when the maximum is unlimited. If a lower value is specified, it will simply be increased to 2049 automatically.

[range] [1, 100 million], <= max_weak_references

See also

DDS_DatabaseQosPolicy::max_weak_references (p. 834)

Categories

Immutable (p. 333), **Middleware Memory Management** (p. 297)

9.29 DDS_DataReaderCacheStatus Struct Reference

<<***extension***>> *(*p. *432)* The status of the reader's cache.

Public Attributes

- **DDS_LongLong sample_count_peak**

  The highest number of samples in the reader's queue over the lifetime of the reader.
- **DDS_LongLong sample_count**

  The number of samples in the reader's queue.
- **DDS_LongLong old_source_timestamp_dropped_sample_count**

  The number of samples dropped as a result of receiving a sample older than the last one, using ***DDS_BY_SOURCE_TIMESTAMP_DESTINATIONORDER_QOS*** *(*p. *585).*
- **DDS_LongLong tolerance_source_timestamp_dropped_sample_count**

  The number of samples dropped as a result of receiving a sample in the future, using ***DDS_BY_SOURCE_TIMESTAMP_DESTINATIONORDER_QOS*** *(*p. *585).*
- **DDS_LongLong ownership_dropped_sample_count**

  The number of samples dropped as a result of receiving a sample from a DataWriter with a lower strength, using Exclusive Ownership.
- **DDS_LongLong content_filter_dropped_sample_count**

  The number of user samples filtered by the DataReader due to Content-Filtered Topics.
- **DDS_LongLong time_based_filter_dropped_sample_count**

  The number of user samples filtered by the DataReader due to ***DDS_TimeBasedFilterQosPolicy*** *(*p. *1389).*
- **DDS_LongLong expired_dropped_sample_count**

  The number of samples expired by the DataReader due to ***DDS_LifespanQosPolicy*** *(*p. *1175) or the autopurge sample delays.*
- **DDS_LongLong virtual_duplicate_dropped_sample_count**

  The number of virtual duplicate samples dropped by the DataReader. A sample is a virtual duplicate if it has the same identity (Virtual Writer GUID and Virtual Sequence Number) as a previously received sample.
- **DDS_LongLong replaced_dropped_sample_count**

  The number of samples replaced by the DataReader due to ***DDS_KEEP_LAST_HISTORY_QOS*** *(*p. *604) replacement.*
- **DDS_LongLong writer_removed_batch_sample_dropped_sample_count**

  The number of batch samples received by the DataReader that were marked as removed by the DataWriter.
- **DDS_LongLong total_samples_dropped_by_instance_replacement**

  The number of samples with sample state ***DDS_NOT_READ_SAMPLE_STATE*** *(*p. *162) that were dropped when removing an instance due to instance replacement. See* ***DDS_DataReaderResourceLimitsQosPolicy::instance_replacement*** *(*p. *883) for more details about when instances are replaced.*
- **DDS_LongLong alive_instance_count**

  The number of instances in the DataReader's queue with an instance state equal to ***DDS_ALIVE_INSTANCE_STATE*** *(*p. *166).*
- **DDS_LongLong alive_instance_count_peak**

  The highest value of ***DDS_DataReaderCacheStatus::alive_instance_count*** *(*p. *840) over the lifetime of the DataReader.*
- **DDS_LongLong no_writers_instance_count**

  The number of instances in the DataReader's queue with an instance state equal to ***DDS_NOT_ALIVE_NO_WRITERS_INSTANCE_STATE*** *(*p. *166).*
- **DDS_LongLong no_writers_instance_count_peak**

  The highest value of ***DDS_DataReaderCacheStatus::no_writers_instance_count*** *(*p. *841) over the lifetime of the DataReader.*

- **DDS_LongLong disposed_instance_count**

  *The number of instances in the DataReader's queue with an instance state equal to **DDS_NOT_ALIVE_DISPOSED_INSTANCE_STATE** (p. 166).*
- **DDS_LongLong disposed_instance_count_peak**

  *The highest value of **DDS_DataReaderCacheStatus::disposed_instance_count** (p. 841) over the lifetime of the DataReader.*
- **DDS_LongLong detached_instance_count**

  The number of minimal instance states currently being maintained in the DataReader's queue.
- **DDS_LongLong detached_instance_count_peak**

  *The highest value of **DDS_DataReaderCacheStatus::detached_instance_count** (p. 842) over the lifetime of the DataReader.*
- **DDS_LongLong compressed_sample_count**

  The number of received samples compressed by a DataWriter.

9.29.1 Detailed Description

<<***extension***>> *(p. 432)* The status of the reader's cache.

Entity:

DDSDataReader (p. 1556)

9.29.2 Member Data Documentation

9.29.2.1 sample_count_peak

```
DDS_LongLong DDS_DataReaderCacheStatus::sample_count_peak
```

The highest number of samples in the reader's queue over the lifetime of the reader.

9.29.2.2 sample_count

```
DDS_LongLong DDS_DataReaderCacheStatus::sample_count
```

The number of samples in the reader's queue.

includes samples that may not yet be available to be read or taken by the user, due to samples being received out of order or **PRESENTATION** (p. 615)

9.29.2.3 old_source_timestamp_dropped_sample_count

DDS_LongLong DDS_DataReaderCacheStatus::old_source_timestamp_dropped_sample_count

The number of samples dropped as a result of receiving a sample older than the last one, using **DDS_BY_SOURCE_TIMESTAMP_DESTINATIONORDER_QOS** (p. 585).

When the DataReader is using **DDS_BY_SOURCE_TIMESTAMP_DESTINATIONORDER_QOS** (p. 585):

- If the DataReader receives a sample for an instance with a source timestamp that is older than the last source timestamp received for the instance, the sample is dropped.
- If the DataReader receives a sample for an instance with a source timestamp that is equal to the last source timestamp received for the instance and the writer has a higher virtual GUID, the sample is dropped.

9.29.2.4 tolerance_source_timestamp_dropped_sample_count

DDS_LongLong DDS_DataReaderCacheStatus::tolerance_source_timestamp_dropped_sample_count

The number of samples dropped as a result of receiving a sample in the future, using **DDS_BY_SOURCE_TIMESTAMP_DESTINATIONORDER_QOS** (p. 585).

When the DataReader is using **DDS_BY_SOURCE_TIMESTAMP_DESTINATIONORDER_QOS** (p. 585): the DataReader will accept a sample only if the source timestamp is no farther in the future from the reception timestamp than the source_timestamp_tolerance. Otherwise, the sample is dropped.

9.29.2.5 ownership_dropped_sample_count

DDS_LongLong DDS_DataReaderCacheStatus::ownership_dropped_sample_count

The number of samples dropped as a result of receiving a sample from a DataWriter with a lower strength, using Exclusive Ownership.

When using Exclusive Ownership, the DataReader receives data from multiple DataWriters. Each instance can only be owned by one DataWriter.

If other DataWriters write samples on this instance, the samples will be dropped.

9.29.2.6 content_filter_dropped_sample_count

DDS_LongLong DDS_DataReaderCacheStatus::content_filter_dropped_sample_count

The number of user samples filtered by the DataReader due to Content-Filtered Topics.

When using a content filter on the DataReader side, if the sample received by the DataReader does not pass the filter, it will be dropped.

9.29.2.7 time_based_filter_dropped_sample_count

DDS_LongLong DDS_DataReaderCacheStatus::time_based_filter_dropped_sample_count

The number of user samples filtered by the DataReader due to **DDS_TimeBasedFilterQosPolicy** (p. 1389).

When using **TIME_BASED_FILTER** (p. 639) on the DataReader side, if the sample received by the DataReader does not pass the minimum_separation filter, it will be dropped.

9.29.2.8 expired_dropped_sample_count

DDS_LongLong DDS_DataReaderCacheStatus::expired_dropped_sample_count

The number of samples expired by the DataReader due to **DDS_LifespanQosPolicy** (p. 1175) or the autopurge sample delays.

- **DDS_LifespanQosPolicy** (p. 1175)
  When a sample expires due to **DDS_LifespanQosPolicy** (p. 1175), the data is removed from the DataReader caches. This sample will be considered dropped if its **DDS_SampleStateKind** (p. 161) was **DDS_NOT_READ↩ _SAMPLE_STATE** (p. 162).
- **DDS_ReaderDataLifecycleQosPolicy::autopurge_nowriter_samples_delay** (p. 1291)
  When a sample expires due to **DDS_ReaderDataLifecycleQosPolicy::autopurge_nowriter_samples_delay** (p. 1291), this sample will be considered dropped if its **DDS_SampleStateKind** (p. 161) was **DDS_NOT_READ↩ _SAMPLE_STATE** (p. 162).
- **DDS_ReaderDataLifecycleQosPolicy::autopurge_disposed_samples_delay** (p. 1291)
  When a sample expires due to **DDS_ReaderDataLifecycleQosPolicy::autopurge_disposed_samples_delay** (p. 1291), this sample will be considered dropped if its **DDS_SampleStateKind** (p. 161) was **DDS_NOT_READ↩ _SAMPLE_STATE** (p. 162).

9.29.2.9 virtual_duplicate_dropped_sample_count

DDS_LongLong DDS_DataReaderCacheStatus::virtual_duplicate_dropped_sample_count

The number of virtual duplicate samples dropped by the DataReader. A sample is a virtual duplicate if it has the same identity (Virtual Writer GUID and Virtual Sequence Number) as a previously received sample.

When two DataWriters with the same logical data source publish a sample with the same sequence_number, one sample will be dropped and the other will be received by the DataReader.

This can happen when multiple writers are writing on behalf of the same original DataWriter: for example, in systems with redundant Routing Services or when a DataReader is receiving samples both directly from the original DataWriter and from an instance of Persistence Service.

9.29.2.10 replaced_dropped_sample_count

DDS_LongLong DDS_DataReaderCacheStatus::replaced_dropped_sample_count

The number of samples replaced by the DataReader due to **DDS_KEEP_LAST_HISTORY_QOS** (p. 604) replacement.

When the number of samples for an instance in the queue reaches the **DDS_HistoryQosPolicy::depth** (p. 1164) value, a new sample for the instance will replace the oldest sample for the instance in the queue.

The new sample will be accepted and the old sample will be dropped.

This counter will only be updated if the replaced sample's **DDS_SampleStateKind** (p. 161) was **DDS_NOT_READ_↩ SAMPLE_STATE** (p. 162).

9.29.2.11 writer_removed_batch_sample_dropped_sample_count

DDS_LongLong DDS_DataReaderCacheStatus::writer_removed_batch_sample_dropped_sample_count

The number of batch samples received by the DataReader that were marked as removed by the DataWriter.

When the DataReader receives a batch, the batch can contain samples marked as removed by the DataWriter. Examples of removed samples in a batch could be because of sample replacement due to **DDS_KEEP_LAST_HISTORY↩ _QOS** (p. 604) **DDS_HistoryQosPolicy** (p. 1162) QoS on the DataWriter or because the duration in **DDS_Lifespan↩ QosPolicy** (p. 1175) was reached. By default, any sample marked as removed from a batch is dropped (unless you set the dds.data_reader.accept_writer_removed_batch_samples property in the **DDS_PropertyQosPolicy** (p. 1260) to true). Note: Historical data with removed batch samples written before the DataReader joined the DDS domain will also be included in the count.

9.29.2.12 total_samples_dropped_by_instance_replacement

DDS_LongLong DDS_DataReaderCacheStatus::total_samples_dropped_by_instance_replacement

The number of samples with sample state **DDS_NOT_READ_SAMPLE_STATE** (p. 162) that were dropped when removing an instance due to instance replacement. See **DDS_DataReaderResourceLimitsQosPolicy::instance_↩ replacement** (p. 883) for more details about when instances are replaced.

9.29.2.13 alive_instance_count

DDS_LongLong DDS_DataReaderCacheStatus::alive_instance_count

The number of instances in the DataReader's queue with an instance state equal to **DDS_ALIVE_INSTANCE_STATE** (p. 166).

9.29.2.14 alive_instance_count_peak

DDS_LongLong DDS_DataReaderCacheStatus::alive_instance_count_peak

The highest value of **DDS_DataReaderCacheStatus::alive_instance_count** (p. 840) over the lifetime of the DataReader.

See also

DDS_DataReaderCacheStatus::alive_instance_count (p. 840)

9.29.2.15 no_writers_instance_count

DDS_LongLong DDS_DataReaderCacheStatus::no_writers_instance_count

The number of instances in the DataReader's queue with an instance state equal to **DDS_NOT_ALIVE_NO_WRITERS_INSTANCE_STATE** (p. 166).

9.29.2.16 no_writers_instance_count_peak

DDS_LongLong DDS_DataReaderCacheStatus::no_writers_instance_count_peak

The highest value of **DDS_DataReaderCacheStatus::no_writers_instance_count** (p. 841) over the lifetime of the DataReader.

See also

DDS_DataReaderCacheStatus::no_writers_instance_count (p. 841)

9.29.2.17 disposed_instance_count

DDS_LongLong DDS_DataReaderCacheStatus::disposed_instance_count

The number of instances in the DataReader's queue with an instance state equal to **DDS_NOT_ALIVE_DISPOSED_INSTANCE_STATE** (p. 166).

9.29.2.18 disposed_instance_count_peak

DDS_LongLong DDS_DataReaderCacheStatus::disposed_instance_count_peak

The highest value of **DDS_DataReaderCacheStatus::disposed_instance_count** (p. 841) over the lifetime of the DataReader.

See also

DDS_DataReaderCacheStatus::disposed_instance_count (p. 841)

9.29.2.19 detached_instance_count

DDS_LongLong DDS_DataReaderCacheStatus::detached_instance_count

The number of minimal instance states currently being maintained in the DataReader's queue.

If **DDS_DataReaderResourceLimitsQosPolicy::keep_minimum_state_for_instances** (p. 881) is true, the Data↩ Reader will keep up to a maximum of **DDS_DataReaderResourceLimitsQosPolicy::max_total_instances** (p. 877) detached instances in its queue. For a more in-depth description of detached instances, refer to **DDS_DataReader↩ ResourceLimitsQosPolicy::max_total_instances** (p. 877).

9.29.2.20 detached_instance_count_peak

DDS_LongLong DDS_DataReaderCacheStatus::detached_instance_count_peak

The highest value of **DDS_DataReaderCacheStatus::detached_instance_count** (p. 842) over the lifetime of the DataReader.

See also

DDS_DataReaderCacheStatus::detached_instance_count (p. 842)

9.29.2.21 compressed_sample_count

DDS_LongLong DDS_DataReaderCacheStatus::compressed_sample_count

The number of received samples compressed by a DataWriter.

These include data, dispose, and unregister samples sent by a DataWriter.

9.30 DDS_DataReaderProtocolQosPolicy Struct Reference

<<***extension***>> *(p. 213)* Along with **DDS_WireProtocolQosPolicy** (p. 1511) and **DDS_DataWriterProtocolQos↩ Policy** (p. 892), this QoS policy configures the DDS on-the-network protocol (RTPS).

Public Attributes

- struct **DDS_GUID_t virtual_guid**

  <<***extension***>> *(p. 213) The virtual GUID (Global Unique Identifier).*
- **DDS_UnsignedLong rtps_object_id**

  <<***extension***>> *(p. 213) The RTPS Object ID.*
- **DDS_Boolean expects_inline_qos**

  [DEPRECATED] *(p. **384**)* <<***extension***>> *(p. 213) Specifies whether this DataReader expects inline QoS with every sample.*
- **DDS_Boolean disable_positive_acks**

  <<***extension***>> *(p. 213) Whether the reader sends positive acknowledgements to writers.*
- **DDS_Boolean propagate_dispose_of_unregistered_instances**

  <<***extension***>> *(p. 213) Indicates whether or not an instance can move to the **DDS_NOT_ALIVE_DISPOSED_↩ INSTANCE_STATE** (p. 166) state without being in the **DDS_ALIVE_INSTANCE_STATE** (p. 166) state.*
- **DDS_Boolean propagate_unregister_of_disposed_instances**

  <<***extension***>> *(p. 213) Indicates whether or not an instance can move to the **DDS_NOT_ALIVE_NO_WRITERS_↩ INSTANCE_STATE** (p. 166) state directly from the **DDS_NOT_ALIVE_DISPOSED_INSTANCE_STATE** (p. 166).*
- struct **DDS_RtpsReliableReaderProtocol_t rtps_reliable_reader**

  <<***extension***>> *(p. 213) RTPS protocol-related configuration settings for the RTPS reliable reader associated with a **DDSDataReader** (p. 1556). This parameter only has effect if the reader is configured with **DDS_RELIABLE_↩ RELIABILITY_QOS** (p. 633) **DDS_ReliabilityQosPolicyKind** (p. 632).*

9.30.1 Detailed Description

<<***extension***>> *(p. 213)* Along with **DDS_WireProtocolQosPolicy** (p. 1511) and **DDS_DataWriterProtocolQos↩ Policy** (p. 892), this QoS policy configures the DDS on-the-network protocol (RTPS).

DDS has a standard protocol for packet (user and meta data) exchange between applications using DDS for communications. This QoS policy and **DDS_DataReaderProtocolQosPolicy** (p. 843) give you control over configurable portions of the protocol, including the configuration of the reliable data delivery mechanism of the protocol on a per DataWriter or DataReader basis.

These configuration parameters control timing, timeouts, and give you the ability to tradeoff between speed of data loss detection and repair versus network and CPU bandwidth used to maintain reliability.

It is important to tune the reliability protocol (on a per **DDSDataWriter** (p. 1589) and **DDSDataReader** (p. 1556) basis) to meet the requirements of the end-user application so that data can be sent between DataWriters and DataReaders in an efficient and optimal manner in the presence of data loss.

You can also use this QoS policy to control how RTI Connext responds to "slow" reliable DataReaders or ones that disconnect or are otherwise lost. See **DDS_ReliabilityQosPolicy** (p. 1295) for more information on the per-Data↩ Reader/DataWriter reliability configuration. **DDS_HistoryQosPolicy** (p. 1162) and **DDS_ResourceLimitsQosPolicy** (p. 1308) also play an important role in the DDS reliable protocol.

This QoS policy is an extension to the DDS standard.

Entity:

DDSDataReader (p. 1556)

Properties:

RxO (p. 556) = N/A
Changeable (p. 556) = **NO** (p. 556)

9.30.2 Member Data Documentation

9.30.2.1 virtual_guid

`struct` **DDS_GUID_t** `DDS_DataReaderProtocolQosPolicy::virtual_guid`

<<***extension***>> *(p. 213)* The virtual GUID (Global Unique Identifier).

The virtual GUID is used to uniquely identify different incarnations of the same **DDSDataReader** (p. 1556).

The association between a **DDSDataReader** (p. 1556) and its persisted state is done using the virtual GUID.

[default] DDS_GUID_AUTO (p. 527)

Categories

Immutable (p. 333), **Discovery Metadata** (p. 377)

9.30.2.2 rtps_object_id

DDS_UnsignedLong `DDS_DataReaderProtocolQosPolicy::rtps_object_id`

<<***extension***>> *(p. 213)* The RTPS Object ID.

This value is used to determine the RTPS object ID of a data reader according to the DDS-RTPS Interoperability Wire Protocol.

Only the last 3 bytes are used; the most significant byte is ignored.

If the default value is specified, RTI Connext will automatically assign the object ID based on a counter value (per participant) starting at 0x00800000. That value is incremented for each new data reader.

A rtps_object_id value in the interval [0x00800000,0x00ffffff] may collide with the automatic values assigned by RTI Connext. In those cases, the recomendation is not to use automatic object ID assigment.

[default] DDS_RTPS_AUTO_ID (p. **??**)

[range] [0,0x00ffffff]

Categories

Immutable (p. 333)

9.30.2.3 expects_inline_qos

```
DDS_Boolean DDS_DataReaderProtocolQosPolicy::expects_inline_qos
```

[DEPRECATED] (p. **384**) <<***extension***>> *(p. 213)* Specifies whether this DataReader expects inline QoS with every sample.

RTI Connext DataWriters do not match with DataReaders that set this field to **DDS_BOOLEAN_TRUE** (p. 513) (because RTI Connext DataWriters do not support sending inline QoS), but here is how the field is meant to be used:

In RTI Connext, a **DDSDataReader** (p. 1556) nominally relies on Discovery to propagate QoS on a matched **DDSDataWriter** (p. 1589).

Alternatively, a **DDSDataReader** (p. 1556) may get information on a matched **DDSDataWriter** (p. 1589) through QoS sent inline with a sample.

Asserting **DDS_DataReaderProtocolQosPolicy::expects_inline_qos** (p. 844) indicates to a matching **DDSDataWriter** (p. 1589) that this **DDSDataReader** (p. 1556) expects to receive inline QoS with every sample. The complete set of inline QoS that a **DDSDataWriter** (p. 1589) may send inline is specified by the Real-Time Publish-Subscribe (RTPS) Wire Interoperability Protocol.

Because RTI Connext **DDSDataWriter** (p. 1589) and **DDSDataReader** (p. 1556) cache Discovery information, inline QoS are largely redundant and thus unecessary. Only for other stateless implementations whose **DDSDataReader** (p. 1556) does not cache Discovery information is inline QoS necessary.

Also note that inline QoS are additional wire-payload that consume additional bandwidth and serialization and deserialization time.

[default] DDS_BOOLEAN_FALSE (p. 513)

Categories

Immutable (p. 333), **Discovery Metadata** (p. 377)

9.30.2.4 disable_positive_acks

```
DDS_Boolean DDS_DataReaderProtocolQosPolicy::disable_positive_acks
```

<<***extension***>> *(p. 213)* Whether the reader sends positive acknowledgements to writers.

If set to **DDS_BOOLEAN_TRUE** (p. 513), the reader does not send positive acknowledgments (ACKs) in response to Heartbeat messages. The reader will send negative acknowledgements (NACKs) when a Heartbeat advertises samples that it has not received.

Otherwise, if set to **DDS_BOOLEAN_FALSE** (p. 513) (the default), the reader will send ACKs to writers that expect ACKs (**DDS_DataWriterProtocolQosPolicy::disable_positive_acks** (p. 894) = **DDS_BOOLEAN_FALSE** (p. 513)) and it will not send ACKs to writers that disable ACKs (**DDS_DataWriterProtocolQosPolicy::disable_positive_acks** (p. 894) = **DDS_BOOLEAN_TRUE** (p. 513))

[default] DDS_BOOLEAN_FALSE (p. 513)

Categories

Immutable (p. 333), **Discovery Metadata** (p. 377), **Reliability** (p. 314)

9.30.2.5 propagate_dispose_of_unregistered_instances

```
DDS_Boolean DDS_DataReaderProtocolQosPolicy::propagate_dispose_of_unregistered_instances
```

<<***extension***>> *(p. 213)* Indicates whether or not an instance can move to the **DDS_NOT_ALIVE_DISPOSED_INSTANCE_STATE** (p. 166) state without being in the **DDS_ALIVE_INSTANCE_STATE** (p. 166) state.

This field only applies to keyed readers.

When the field is set to **DDS_BOOLEAN_TRUE** (p. 513), the DataReader will receive dispose notifications even if the instance is not alive.

To guarantee the key availability through the usage of the API **FooDataReader::get_key_value** (p. 1941), this option should be used in combination with setting **DDS_DataWriterProtocolQosPolicy::serialize_key_with_dispose** (p. 895) on the DataWriter to **DDS_BOOLEAN_TRUE** (p. 513).

[default] DDS_BOOLEAN_FALSE (p. 513)

Categories

Immutable (p. 333), **Instance Lifecycle** (p. 286)

9.30.2.6 propagate_unregister_of_disposed_instances

```
DDS_Boolean DDS_DataReaderProtocolQosPolicy::propagate_unregister_of_disposed_instances
```

<<***extension***>> *(p. 213)* Indicates whether or not an instance can move to the **DDS_NOT_ALIVE_NO_WRITERS_INSTANCE_STATE** (p. 166) state directly from the **DDS_NOT_ALIVE_DISPOSED_INSTANCE_STATE** (p. 166).

This field only applies to keyed readers.

When the field is set to **DDS_BOOLEAN_TRUE** (p. 513), the DataReader will receive unregister notifications even if the instance is not alive.

[default] DDS_BOOLEAN_FALSE (p. 513)

Categories

Immutable (p. 333), **Instance Lifecycle** (p. 286)

9.30.2.7 rtps_reliable_reader

`struct` **`DDS_RtpsReliableReaderProtocol_t`** `DDS_DataReaderProtocolQosPolicy::rtps_reliable_reader`

<<***extension***>> *(p. 213)* RTPS protocol-related configuration settings for the RTPS reliable reader associated with a **DDSDataReader** (p. 1556). This parameter only has effect if the reader is configured with **DDS_RELIABLE_↩ RELIABILITY_QOS** (p. 633) **DDS_ReliabilityQosPolicyKind** (p. 632).

For details, refer to the **DDS_RtpsReliableReaderProtocol_t** (p. 1314)

[default] See **DDS_RtpsReliableReaderProtocol_t** (p. 1314)

Categories

Immutable (p. 333), **Reliability** (p. 314)

9.31 DDS_DataReaderProtocolStatus Struct Reference

<<***extension***>> *(p. 432)* The status of a reader's internal protocol related metrics, like the number of samples received, filtered, rejected; and status of wire protocol traffic.

Public Attributes

- **DDS_LongLong received_sample_count**

  The number of samples received by a DataReader.
- **DDS_LongLong received_sample_count_change**

  *The change in **DDS_DataReaderProtocolStatus::received_sample_count** (p. 849) since the last time the status was read.*
- **DDS_LongLong received_sample_bytes**

  The number of bytes received by a DataReader.
- **DDS_LongLong received_sample_bytes_change**

  *The change in **DDS_DataReaderProtocolStatus::received_sample_bytes** (p. 850) since the last time the status was read.*
- **DDS_LongLong duplicate_sample_count**

  The number of samples from a remote DataWriter received, not for the first time, by a local DataReader.
- **DDS_LongLong duplicate_sample_count_change**

  *The change in **DDS_DataReaderProtocolStatus::duplicate_sample_count** (p. 851) since the last time the status was read.*
- **DDS_LongLong duplicate_sample_bytes**

  The number of bytes of samples from a remote DataWriter received, not for the first time, by a local DataReader.
- **DDS_LongLong duplicate_sample_bytes_change**

  *The change in **DDS_DataReaderProtocolStatus::duplicate_sample_bytes** (p. 851) since the last time the status was read.*
- **DDS_LongLong filtered_sample_count**

  ***[DEPRECATED]**. See: **DDS_DataReaderCacheStatus::time_based_filter_dropped_sample_count** (p. 838) **DDS_↩ DataReaderCacheStatus::content_filter_dropped_sample_count** (p. 838)*
- **DDS_LongLong filtered_sample_count_change**

[DEPRECATED]. See: ***DDS_DataReaderCacheStatus::time_based_filter_dropped_sample_count*** *(p. 838)* ***DDS_↩ DataReaderCacheStatus::content_filter_dropped_sample_count*** *(p. 838)*

- **DDS_LongLong filtered_sample_bytes**

  [DEPRECATED]. See: ***DDS_DataReaderCacheStatus::time_based_filter_dropped_sample_count*** *(p. 838)* ***DDS_↩ DataReaderCacheStatus::content_filter_dropped_sample_count*** *(p. 838)*

- **DDS_LongLong filtered_sample_bytes_change**

  [DEPRECATED]. See: ***DDS_DataReaderCacheStatus::time_based_filter_dropped_sample_count*** *(p. 838)* ***DDS_↩ DataReaderCacheStatus::content_filter_dropped_sample_count*** *(p. 838)*

- **DDS_LongLong received_heartbeat_count**

  The number of Heartbeats from a remote DataWriter received by a local DataReader.

- **DDS_LongLong received_heartbeat_count_change**

  The change in ***DDS_DataReaderProtocolStatus::received_heartbeat_count*** *(p. 852) since the last time the status was read.*

- **DDS_LongLong received_heartbeat_bytes**

  The number of bytes of Heartbeats from a remote DataWriter received by a local DataReader.

- **DDS_LongLong received_heartbeat_bytes_change**

  The change in ***DDS_DataReaderProtocolStatus::received_heartbeat_bytes*** *(p. 853) since the last time the status was read.*

- **DDS_LongLong sent_ack_count**

  The number of ACKs sent from a local DataReader to a matching remote DataWriter.

- **DDS_LongLong sent_ack_count_change**

  The change in ***DDS_DataReaderProtocolStatus::sent_ack_count*** *(p. 853) since the last time the status was read.*

- **DDS_LongLong sent_ack_bytes**

  The number of bytes of ACKs sent from a local DataReader to a matching remote DataWriter.

- **DDS_LongLong sent_ack_bytes_change**

  The change in ***DDS_DataReaderProtocolStatus::sent_ack_bytes*** *(p. 853) since the last time the status was read.*

- **DDS_LongLong sent_nack_count**

  The number of NACKs sent from a local DataReader to a matching remote DataWriter.

- **DDS_LongLong sent_nack_count_change**

  The change in ***DDS_DataReaderProtocolStatus::sent_nack_count*** *(p. 854) since the last time the status was read.*

- **DDS_LongLong sent_nack_bytes**

  The number of bytes of NACKs sent from a local DataReader to a matching remote DataWriter.

- **DDS_LongLong sent_nack_bytes_change**

  The change in ***DDS_DataReaderProtocolStatus::sent_nack_bytes*** *(p. 854) since the last time the status was read.*

- **DDS_LongLong received_gap_count**

  The number of GAPs received from remote DataWriter to this DataReader.

- **DDS_LongLong received_gap_count_change**

  The change in ***DDS_DataReaderProtocolStatus::received_gap_count*** *(p. 854) since the last time the status was read.*

- **DDS_LongLong received_gap_bytes**

  The number of bytes of GAPs received from remote DataWriter to this DataReader.

- **DDS_LongLong received_gap_bytes_change**

  The change in ***DDS_DataReaderProtocolStatus::received_gap_bytes*** *(p. 855) since the last time the status was read.*

- **DDS_LongLong rejected_sample_count**

  The number of times a sample is rejected because it cannot be accepted by a reliable DataReader.

- **DDS_LongLong rejected_sample_count_change**

  The change in ***DDS_DataReaderProtocolStatus::rejected_sample_count*** *(p. 855) since the last time the status was read.*

- struct **DDS_SequenceNumber_t first_available_sample_sequence_number**

Sequence number of the first available sample in a matched DataWriters reliability queue.

- struct **DDS_SequenceNumber_t last_available_sample_sequence_number**

  Sequence number of the last available sample in a matched Datawriter's reliability queue.
- struct **DDS_SequenceNumber_t last_committed_sample_sequence_number**

  Sequence number of the newest sample received from the matched DataWriter committed to the DataReader's queue.
- **DDS_Long uncommitted_sample_count**

  Number of received samples that are not yet available to be read or taken, due to being received out of order.
- **DDS_LongLong out_of_range_rejected_sample_count**

  The number of samples dropped by the DataReader due to received window is full and the sample is out-of-order.
- **DDS_LongLong received_fragment_count**

  The number of DATA_FRAG messages that have been received by this DataReader.
- **DDS_LongLong dropped_fragment_count**

  The number of DATA_FRAG messages that have been dropped by a DataReader.
- **DDS_LongLong reassembled_sample_count**

  The number of fragmented samples that have been reassembled by a DataReader.
- **DDS_LongLong sent_nack_fragment_count**

  The number of NACK fragments that have been sent from a DataReader to a DataWriter.
- **DDS_LongLong sent_nack_fragment_bytes**

  The number of NACK fragment bytes that have been sent from a DataReader to a DataWriter.

9.31.1 Detailed Description

<<***extension***>> *(p. 432)* The status of a reader's internal protocol related metrics, like the number of samples received, filtered, rejected; and status of wire protocol traffic.

Entity:

DDSDataReader (p. 1556)

9.31.2 Member Data Documentation

9.31.2.1 received_sample_count

```
DDS_LongLong DDS_DataReaderProtocolStatus::received_sample_count
```

The number of samples received by a DataReader.

Depending on how the **DDS_DataReaderProtocolStatus** (p. 847) was obtained this may count samples coming from a specific DataWriter or from all the DataWriters that are matched with the DataReader.

If the **DDS_DataReaderProtocolStatus** (p. 847) is obtained using the **DDSDataReader::get_datareader_protocol_status** (p. 1573) operation then it will count samples from any DataWriter. If the DataReaderProtocolStatus is obtained using the **DDSDataReader::get_matched_publication_datareader_protocol_status** (p. 1573) then it will count the samples for the DataWriter specified as a parameter to the function.

Duplicate samples arriving from the DataWriter(s) (e.g. via multiple network paths) are detected prior to increasing this counter. The duplicate samples are counted by **DDS_DataReaderProtocolStatus::duplicate_sample_count** (p. 851).

If the DataReader has specified a ContentFilter the received samples that do not pass the filter are part of this counter. The filtered samples are counted by **DDS_DataReaderProtocolStatus::filtered_sample_count** (p. 851).

Samples rejected because they do not fit on the DataReader Queue are also part of this counter.

Note the received_sample_count counts samples received from all DataWriters and it does not necessarily match the number of samples accepted into the DataReader Queue. This is because:

- Samples can also be inserted into the DataReader Queue by lifecycle events that are locally detected like an instance becoming not alive as a result of DataWriters leaving the network.
- Samples can be filtered out due to ContentFilter or TimeFilter
- Samples can be rejected because there is no space in DataReader Queue

Note that when data is fragmented, this statistic is updated when all of the fragments required to reassemble a sample are received, not when individual fragments are received.

9.31.2.2 received_sample_count_change

```
DDS_LongLong DDS_DataReaderProtocolStatus::received_sample_count_change
```

The change in **DDS_DataReaderProtocolStatus::received_sample_count** (p. 849) since the last time the status was read.

See also

DDS_DataReaderProtocolStatus::received_sample_count (p. 849)

Note that when data is fragmented, this statistic is updated when all of the fragments required to reassemble a sample are received, not when individual fragments are received.

9.31.2.3 received_sample_bytes

```
DDS_LongLong DDS_DataReaderProtocolStatus::received_sample_bytes
```

The number of bytes received by a DataReader.

See also

DDS_DataReaderProtocolStatus::received_sample_count (p. 849)

Note that when data is fragmented, this statistic is updated upon the receipt of each fragment, not when a sample is reassembled.

9.31.2.4 received_sample_bytes_change

DDS_LongLong DDS_DataReaderProtocolStatus::received_sample_bytes_change

The change in **DDS_DataReaderProtocolStatus::received_sample_bytes** (p. 850) since the last time the status was read.

See also

DDS_DataReaderProtocolStatus::received_sample_count_change (p. 850)

Note that when data is fragmented, this statistic is updated upon the receipt of each fragment, not when a sample is reassembled.

9.31.2.5 duplicate_sample_count

DDS_LongLong DDS_DataReaderProtocolStatus::duplicate_sample_count

The number of samples from a remote DataWriter received, not for the first time, by a local DataReader.

Such samples can be redundant, out-of-order, etc. and are not stored in the reader's queue.

9.31.2.6 duplicate_sample_count_change

DDS_LongLong DDS_DataReaderProtocolStatus::duplicate_sample_count_change

The change in **DDS_DataReaderProtocolStatus::duplicate_sample_count** (p. 851) since the last time the status was read.

Such samples can be redundant, out-of-order, etc. and are not stored in the reader's queue.

9.31.2.7 duplicate_sample_bytes

DDS_LongLong DDS_DataReaderProtocolStatus::duplicate_sample_bytes

The number of bytes of samples from a remote DataWriter received, not for the first time, by a local DataReader.

Such samples can be redundant, out-of-order, etc. and are not stored in the reader's queue.

9.31.2.8 duplicate_sample_bytes_change

DDS_LongLong DDS_DataReaderProtocolStatus::duplicate_sample_bytes_change

The change in **DDS_DataReaderProtocolStatus::duplicate_sample_bytes** (p. 851) since the last time the status was read.

Such samples can be redundant, out-of-order, etc. and are not stored in the reader's queue.

9.31.2.9 filtered_sample_count

```
DDS_LongLong DDS_DataReaderProtocolStatus::filtered_sample_count
```

[DEPRECATED]. See: **DDS_DataReaderCacheStatus::time_based_filter_dropped_sample_count** (p. 838) **DDS_DataReaderCacheStatus::content_filter_dropped_sample_count** (p. 838)

9.31.2.10 filtered_sample_count_change

```
DDS_LongLong DDS_DataReaderProtocolStatus::filtered_sample_count_change
```

[DEPRECATED]. See: **DDS_DataReaderCacheStatus::time_based_filter_dropped_sample_count** (p. 838) **DDS_DataReaderCacheStatus::content_filter_dropped_sample_count** (p. 838)

9.31.2.11 filtered_sample_bytes

```
DDS_LongLong DDS_DataReaderProtocolStatus::filtered_sample_bytes
```

[DEPRECATED]. See: **DDS_DataReaderCacheStatus::time_based_filter_dropped_sample_count** (p. 838) **DDS_DataReaderCacheStatus::content_filter_dropped_sample_count** (p. 838)

9.31.2.12 filtered_sample_bytes_change

```
DDS_LongLong DDS_DataReaderProtocolStatus::filtered_sample_bytes_change
```

[DEPRECATED]. See: **DDS_DataReaderCacheStatus::time_based_filter_dropped_sample_count** (p. 838) **DDS_DataReaderCacheStatus::content_filter_dropped_sample_count** (p. 838)

9.31.2.13 received_heartbeat_count

```
DDS_LongLong DDS_DataReaderProtocolStatus::received_heartbeat_count
```

The number of Heartbeats from a remote DataWriter received by a local DataReader.

9.31.2.14 received_heartbeat_count_change

DDS_LongLong DDS_DataReaderProtocolStatus::received_heartbeat_count_change

The change in **DDS_DataReaderProtocolStatus::received_heartbeat_count** (p. 852) since the last time the status was read.

9.31.2.15 received_heartbeat_bytes

DDS_LongLong DDS_DataReaderProtocolStatus::received_heartbeat_bytes

The number of bytes of Heartbeats from a remote DataWriter received by a local DataReader.

9.31.2.16 received_heartbeat_bytes_change

DDS_LongLong DDS_DataReaderProtocolStatus::received_heartbeat_bytes_change

The change in **DDS_DataReaderProtocolStatus::received_heartbeat_bytes** (p. 853) since the last time the status was read.

9.31.2.17 sent_ack_count

DDS_LongLong DDS_DataReaderProtocolStatus::sent_ack_count

The number of ACKs sent from a local DataReader to a matching remote DataWriter.

9.31.2.18 sent_ack_count_change

DDS_LongLong DDS_DataReaderProtocolStatus::sent_ack_count_change

The change in **DDS_DataReaderProtocolStatus::sent_ack_count** (p. 853) since the last time the status was read.

9.31.2.19 sent_ack_bytes

DDS_LongLong DDS_DataReaderProtocolStatus::sent_ack_bytes

The number of bytes of ACKs sent from a local DataReader to a matching remote DataWriter.

9.31.2.20 sent_ack_bytes_change

DDS_LongLong DDS_DataReaderProtocolStatus::sent_ack_bytes_change

The change in **DDS_DataReaderProtocolStatus::sent_ack_bytes** (p. 853) since the last time the status was read.

9.31.2.21 sent_nack_count

DDS_LongLong DDS_DataReaderProtocolStatus::sent_nack_count

The number of NACKs sent from a local DataReader to a matching remote DataWriter.

9.31.2.22 sent_nack_count_change

DDS_LongLong DDS_DataReaderProtocolStatus::sent_nack_count_change

The change in **DDS_DataReaderProtocolStatus::sent_nack_count** (p. 854) since the last time the status was read.

9.31.2.23 sent_nack_bytes

DDS_LongLong DDS_DataReaderProtocolStatus::sent_nack_bytes

The number of bytes of NACKs sent from a local DataReader to a matching remote DataWriter.

9.31.2.24 sent_nack_bytes_change

DDS_LongLong DDS_DataReaderProtocolStatus::sent_nack_bytes_change

The change in **DDS_DataReaderProtocolStatus::sent_nack_bytes** (p. 854) since the last time the status was read.

9.31.2.25 received_gap_count

DDS_LongLong DDS_DataReaderProtocolStatus::received_gap_count

The number of GAPs received from remote DataWriter to this DataReader.

9.31.2.26 received_gap_count_change

DDS_LongLong DDS_DataReaderProtocolStatus::received_gap_count_change

The change in **DDS_DataReaderProtocolStatus::received_gap_count** (p. 854) since the last time the status was read.

9.31.2.27 received_gap_bytes

DDS_LongLong DDS_DataReaderProtocolStatus::received_gap_bytes

The number of bytes of GAPs received from remote DataWriter to this DataReader.

9.31.2.28 received_gap_bytes_change

DDS_LongLong DDS_DataReaderProtocolStatus::received_gap_bytes_change

The change in **DDS_DataReaderProtocolStatus::received_gap_bytes** (p. 855) since the last time the status was read.

9.31.2.29 rejected_sample_count

DDS_LongLong DDS_DataReaderProtocolStatus::rejected_sample_count

The number of times a sample is rejected because it cannot be accepted by a reliable DataReader.

Samples rejected by a reliable DataReader will be NACKed, and they will have to be resent by the DataWriter if they are still available in the DataWriter queue.

This counter will always be 0 when using **DDS_BEST_EFFORT_RELIABILITY_QOS** (p. 633).

9.31.2.30 rejected_sample_count_change

DDS_LongLong DDS_DataReaderProtocolStatus::rejected_sample_count_change

The change in **DDS_DataReaderProtocolStatus::rejected_sample_count** (p. 855) since the last time the status was read.

This counter will always be 0 when using **DDS_BEST_EFFORT_RELIABILITY_QOS** (p. 633).

9.31.2.31 first_available_sample_sequence_number

struct **DDS_SequenceNumber_t** DDS_DataReaderProtocolStatus::first_available_sample_sequence_number

Sequence number of the first available sample in a matched DataWriters reliability queue.

Applicable only for reliable DataReaders, and when retrieving matched DataWriter statuses.

Updated upon receiving Heartbeat submessages from a matched reliable DataWriter.

9.31.2.32 last_available_sample_sequence_number

struct **DDS_SequenceNumber_t** DDS_DataReaderProtocolStatus::last_available_sample_sequence_number

Sequence number of the last available sample in a matched Datawriter's reliability queue.

Applicable only for reliable DataReaders, and when retrieving matched DataWriter statuses.

Updated upon receiving Heartbeat submessages from a matched reliable DataWriter.

9.31.2.33 last_committed_sample_sequence_number

struct **DDS_SequenceNumber_t** DDS_DataReaderProtocolStatus::last_committed_sample_sequence_number

Sequence number of the newest sample received from the matched DataWriter committed to the DataReader's queue.

Applicable only when retrieving matched DataWriter statuses.

For best-effort DataReaders, this is the sequence number of the latest sample received.

For reliable DataReaders, this is the sequence number of the latest sample that is available to be read or taken from the DataReader's queue.

9.31.2.34 uncommitted_sample_count

DDS_Long DDS_DataReaderProtocolStatus::uncommitted_sample_count

Number of received samples that are not yet available to be read or taken, due to being received out of order.

Applicable only when retrieving matched DataWriter statuses.

9.31.2.35 out_of_range_rejected_sample_count

DDS_LongLong DDS_DataReaderProtocolStatus::out_of_range_rejected_sample_count

The number of samples dropped by the DataReader due to received window is full and the sample is out-of-order.

When using **DDS_RELIABLE_RELIABILITY_QOS** (p. 633); if the DataReader received samples out-of-order, they are stored internally until the missing samples are received. The number of out-of-order samples that the DataReader can keep is set by **DDS_RtpsReliableReaderProtocol_t::receive_window_size** (p. 1317). When the received window is full any out-of-order sample received will be dropped.

9.31.2.36 received_fragment_count

DDS_LongLong DDS_DataReaderProtocolStatus::received_fragment_count

The number of DATA_FRAG messages that have been received by this DataReader.

This statistic is incremented upon the receipt of each DATA_FRAG message. Fragments from duplicate samples do not count towards this statistic. Applicable only when data is fragmented.

9.31.2.37 dropped_fragment_count

DDS_LongLong DDS_DataReaderProtocolStatus::dropped_fragment_count

The number of DATA_FRAG messages that have been dropped by a DataReader.

This statistic does not include malformed fragments. Applicable only when data is fragmented.

9.31.2.38 reassembled_sample_count

DDS_LongLong DDS_DataReaderProtocolStatus::reassembled_sample_count

The number of fragmented samples that have been reassembled by a DataReader.

This statistic is incremented when all of the fragments which are required to reassemble an entire sample have been received. Applicable only when data is fragmented.

9.31.2.39 sent_nack_fragment_count

DDS_LongLong DDS_DataReaderProtocolStatus::sent_nack_fragment_count

The number of NACK fragments that have been sent from a DataReader to a DataWriter.

NACK FRAG RTPS messages are sent when large data is used in conjunction with reliable communication. They have the same properties as NACK messages, but instead of applying to samples, they apply to fragments. Applicable only when data is fragmented.

9.31.2.40 sent_nack_fragment_bytes

```
DDS_LongLong DDS_DataReaderProtocolStatus::sent_nack_fragment_bytes
```

The number of NACK fragment bytes that have been sent from a DataReader to a DataWriter.

NACK FRAG RTPS messages are sent when large data is used in conjunction with reliable communication. They have the same properties as NACK messages, but instead of applying to samples, they apply to fragments. Applicable only when data is fragmented.

9.32 DDS_DataReaderQos Struct Reference

QoS policies supported by a **DDSDataReader** (p. 1556) entity.

Public Member Functions

- bool **operator==** (const **DDS_DataReaderQos** &r) const

  Compares two DataReaderQos objects for equality.
- bool **operator!=** (const **DDS_DataReaderQos** &r) const

  Compares two DataReaderQos objects for inequality.
- **DDS_ReturnCode_t print** () const

  *Prints this **DDS_DataReaderQos** (p. 858) to stdout.*
- **DDS_ReturnCode_t to_string** (char *string, **DDS_UnsignedLong** &string_size) const

  *Obtains a string representation of this **DDS_DataReaderQos** (p. 858).*
- **DDS_ReturnCode_t to_string** (char *string, **DDS_UnsignedLong** &string_size, const **DDS_DataReaderQos** &base) const

  *Obtains a string representation of this **DDS_DataReaderQos** (p. 858).*
- **DDS_ReturnCode_t to_string** (char *string, **DDS_UnsignedLong** &string_size, const **DDS_QosPrintFormat** &format) const

  *Obtains a string representation of this **DDS_DataReaderQos** (p. 858).*
- **DDS_ReturnCode_t to_string** (char *string, **DDS_UnsignedLong** &string_size, const **DDS_DataReaderQos** &base, const **DDS_QosPrintFormat** &format) const

  *Obtains a string representation of this **DDS_DataReaderQos** (p. 858).*
- **DDS_ReturnCode_t to_string** (char *string, **DDS_UnsignedLong** &string_size, const **DDS_QosPrintAll_t** &) const

  *Obtains a string representation of this **DDS_DataReaderQos** (p. 858).*
- **DDS_ReturnCode_t to_string** (char *string, **DDS_UnsignedLong** &string_size, const **DDS_QosPrintAll_t** &, const **DDS_QosPrintFormat** &format) const

  *Obtains a string representation of this **DDS_DataReaderQos** (p. 858).*

Public Attributes

- struct **DDS_DurabilityQosPolicy durability**

  *Durability policy, **DURABILITY** (p. 596).*
- struct **DDS_DeadlineQosPolicy deadline**

  *<<**basic**>> (p. 201) Deadline policy, **DEADLINE** (p. 583).*
- struct **DDS_LatencyBudgetQosPolicy latency_budget**

  *Latency budget policy, **LATENCY_BUDGET** (p. 605).*
- struct **DDS_LivelinessQosPolicy liveliness**

  *Liveliness policy, **LIVELINESS** (p. 607).*
- struct **DDS_ReliabilityQosPolicy reliability**

  *Reliability policy, **RELIABILITY** (p. 631).*
- struct **DDS_DestinationOrderQosPolicy destination_order**

  *Destination order policy, **DESTINATION_ORDER** (p. 584).*
- struct **DDS_HistoryQosPolicy history**

  *<<**basic**>> (p. 201) History policy, **HISTORY** (p. 603).*
- struct **DDS_ResourceLimitsQosPolicy resource_limits**

  *Resource limits policy, **RESOURCE_LIMITS** (p. 635).*
- struct **DDS_UserDataQosPolicy user_data**

  *User data policy, **USER_DATA** (p. 653).*
- struct **DDS_OwnershipQosPolicy ownership**

  *<<**basic**>> (p. 201) Ownership policy, **OWNERSHIP** (p. 612).*
- struct **DDS_TimeBasedFilterQosPolicy time_based_filter**

  *<<**basic**>> (p. 201) Time-based filter policy, **TIME_BASED_FILTER** (p. 639)*
- struct **DDS_ReaderDataLifecycleQosPolicy reader_data_lifecycle**

  *Reader data lifecycle policy, **READER_DATA_LIFECYCLE** (p. 630).*
- struct **DDS_DataRepresentationQosPolicy representation**

  *Data representation policy, **DATA_REPRESENTATION** (p. 567).*
- struct **DDS_TypeConsistencyEnforcementQosPolicy type_consistency**

  *Type consistency enforcement policy, **TYPE_CONSISTENCY_ENFORCEMENT** (p. 649).*
- **DDS_DataTagQosPolicy data_tags**

  *DataTag policy, **DATA_TAG** (p. 574).*
- struct **DDS_DataReaderResourceLimitsQosPolicy reader_resource_limits**

  *<<**extension**>> (p. 432) **DDSDataReader** (p. 1556) resource limits policy, **DATA_READER_RESOURCE_LIMITS** (p. 564). This policy is an extension to the DDS standard.*
- struct **DDS_DataReaderProtocolQosPolicy protocol**

  *<<**extension**>> (p. 432) **DDSDataReader** (p. 1556) protocol policy, **DATA_READER_PROTOCOL** (p. 564)*
- struct **DDS_TransportSelectionQosPolicy transport_selection**

  *<<**extension**>> (p. 432) Transport selection policy, **TRANSPORT_SELECTION** (p. 648).*
- struct **DDS_TransportUnicastQosPolicy unicast**

  *<<**extension**>> (p. 432) Unicast transport policy, **TRANSPORT_UNICAST** (p. 649).*
- struct **DDS_TransportMulticastQosPolicy multicast**

  *<<**extension**>> (p. 432) Multicast transport policy, **TRANSPORT_MULTICAST** (p. 645).*
- struct **DDS_PropertyQosPolicy property**

  *<<**basic**>> (p. 201) <<**extension**>> (p. 432) Property policy, **PROPERTY** (p. 617). See also `Property Reference Guide`.*
- struct **DDS_ServiceQosPolicy service**

  *<<**extension**>> (p. 432) Service policy, **SERVICE** (p. 636).*

- struct **DDS_AvailabilityQosPolicy availability**

  <<***extension***>> *(p. 432) Availability policy,* ***AVAILABILITY*** *(p. 561).*
- struct **DDS_EntityNameQosPolicy subscription_name**

  <<***extension***>> *(p. 432) EntityName policy,* ***ENTITY_NAME*** *(p. 601).*
- struct **DDS_TransportPriorityQosPolicy transport_priority**

  Transport priority policy, ***TRANSPORT_PRIORITY*** *(p. 647).*
- struct **DDS_TypeSupportQosPolicy type_support**

  <<***extension***>> *(p. 432) type support data,* ***TYPESUPPORT*** *(p. 651).*

9.32.1 Detailed Description

QoS policies supported by a **DDSDataReader** (p. 1556) entity.

You must set certain members in a consistent manner:

DDS_DataReaderQos::deadline (p. 861) .period >= **DDS_DataReaderQos::time_based_filter** (p. 863) .minimum↩
_separation

DDS_DataReaderQos::history (p. 862) .depth <= **DDS_DataReaderQos::resource_limits** (p. 862) .max_samples↩
_per_instance

DDS_DataReaderQos::resource_limits (p. 862) .max_samples_per_instance <= **DDS_DataReaderQos↩
::resource_limits** (p. 862) .max_samples **DDS_DataReaderQos::resource_limits** (p. 862) .initial_samples <= **DDS↩
_DataReaderQos::resource_limits** (p. 862) .max_samples

DDS_DataReaderQos::resource_limits (p. 862) .initial_instances <= **DDS_DataReaderQos::resource_limits** (p. 862) .max_instances

DDS_DataReaderQos::reader_resource_limits (p. 864) .initial_remote_writers_per_instance <= **DDS_Data↩
ReaderQos::reader_resource_limits** (p. 864) .max_remote_writers_per_instance

DDS_DataReaderQos::reader_resource_limits (p. 864) .initial_infos <= **DDS_DataReaderQos::reader_resource↩
_limits** (p. 864) .max_infos

DDS_DataReaderQos::reader_resource_limits (p. 864) .max_remote_writers_per_instance <= **DDS_DataReader↩
Qos::reader_resource_limits** (p. 864) .max_remote_writers

DDS_DataReaderQos::reader_resource_limits (p. 864) .max_samples_per_remote_writer <= **DDS_DataReader↩
Qos::resource_limits** (p. 862) .max_samples

length of **DDS_DataReaderQos::user_data** (p. 862) .value <= **DDS_DomainParticipantQos::resource_limits** (p. 974) .reader_user_data_max_length

If any of the above are not true, **DDSDataReader::set_qos** (p. 1574) and **DDSDataReader::set_qos_with_profile** (p. 1575) will fail with **DDS_RETCODE_INCONSISTENT_POLICY** (p. 533) and **DDSSubscriber::create_datareader** (p. 1867) will return NULL.

9.32.2 Member Function Documentation

9.32.2.1 operator==()

```
bool DDS_DataReaderQos::operator== (
            const  DDS_DataReaderQos & r ) const  [inline]
```

Compares two DataReaderQos objects for equality.

See also

DDS_DataReaderQos_equals (p. 146)

References **DDS_DataReaderQos_equals()**.

9.32.2.2 operator"!=()

```
bool DDS_DataReaderQos::operator!= (
            const  DDS_DataReaderQos & r ) const  [inline]
```

Compares two DataReaderQos objects for inequality.

See also

DDS_DataReaderQos_equals (p. 146)

References **DDS_DataReaderQos_equals()**.

9.32.3 Member Data Documentation

9.32.3.1 durability

```
struct  DDS_DurabilityQosPolicy DDS_DataReaderQos::durability
```

Durability policy, **DURABILITY** (p. 596).

9.32.3.2 deadline

```
struct  DDS_DeadlineQosPolicy DDS_DataReaderQos::deadline
```

<<***basic***>> *(p. 201)* Deadline policy, **DEADLINE** (p. 583).

9.32.3.3 latency_budget

`struct` **DDS_LatencyBudgetQosPolicy** `DDS_DataReaderQos::latency_budget`

Latency budget policy, **LATENCY_BUDGET** (p. 605).

9.32.3.4 liveliness

`struct` **DDS_LivelinessQosPolicy** `DDS_DataReaderQos::liveliness`

Liveliness policy, **LIVELINESS** (p. 607).

9.32.3.5 reliability

`struct` **DDS_ReliabilityQosPolicy** `DDS_DataReaderQos::reliability`

Reliability policy, **RELIABILITY** (p. 631).

9.32.3.6 destination_order

`struct` **DDS_DestinationOrderQosPolicy** `DDS_DataReaderQos::destination_order`

Destination order policy, **DESTINATION_ORDER** (p. 584).

9.32.3.7 history

`struct` **DDS_HistoryQosPolicy** `DDS_DataReaderQos::history`

<<***basic***>> *(p. 201)* History policy, **HISTORY** (p. 603).

9.32.3.8 resource_limits

`struct` **DDS_ResourceLimitsQosPolicy** `DDS_DataReaderQos::resource_limits`

Resource limits policy, **RESOURCE_LIMITS** (p. 635).

9.32.3.9 user_data

`struct` **DDS_UserDataQosPolicy** DDS_DataReaderQos::user_data

User data policy, **USER_DATA** (p. 653).

9.32.3.10 ownership

`struct` **DDS_OwnershipQosPolicy** DDS_DataReaderQos::ownership

<<***basic***>> *(p. 201)* Ownership policy, **OWNERSHIP** (p. 612).

9.32.3.11 time_based_filter

`struct` **DDS_TimeBasedFilterQosPolicy** DDS_DataReaderQos::time_based_filter

<<***basic***>> *(p. 201)* Time-based filter policy, **TIME_BASED_FILTER** (p. 639)

9.32.3.12 reader_data_lifecycle

`struct` **DDS_ReaderDataLifecycleQosPolicy** DDS_DataReaderQos::reader_data_lifecycle

Reader data lifecycle policy, **READER_DATA_LIFECYCLE** (p. 630).

9.32.3.13 representation

`struct` **DDS_DataRepresentationQosPolicy** DDS_DataReaderQos::representation

Data representation policy, **DATA_REPRESENTATION** (p. 567).

9.32.3.14 type_consistency

`struct` **DDS_TypeConsistencyEnforcementQosPolicy** DDS_DataReaderQos::type_consistency

Type consistency enforcement policy, **TYPE_CONSISTENCY_ENFORCEMENT** (p. 649).

9.32.3.15 data_tags

DDS_DataTagQosPolicy DDS_DataReaderQos::data_tags

DataTag policy, **DATA_TAG** (p. 574).

9.32.3.16 reader_resource_limits

struct **DDS_DataReaderResourceLimitsQosPolicy** DDS_DataReaderQos::reader_resource_limits

<<***extension***>> *(p. 432)* **DDSDataReader** (p. 1556) resource limits policy, **DATA_READER_RESOURCE_LIMITS** (p. 564). This policy is an extension to the DDS standard.

9.32.3.17 protocol

struct **DDS_DataReaderProtocolQosPolicy** DDS_DataReaderQos::protocol

<<***extension***>> *(p. 432)* **DDSDataReader** (p. 1556) protocol policy, **DATA_READER_PROTOCOL** (p. 564)

9.32.3.18 transport_selection

struct **DDS_TransportSelectionQosPolicy** DDS_DataReaderQos::transport_selection

<<***extension***>> *(p. 432)* Transport selection policy, **TRANSPORT_SELECTION** (p. 648).

Specifies the transports available for use by the **DDSDataReader** (p. 1556).

9.32.3.19 unicast

struct **DDS_TransportUnicastQosPolicy** DDS_DataReaderQos::unicast

<<***extension***>> *(p. 432)* Unicast transport policy, **TRANSPORT_UNICAST** (p. 649).

Specifies the unicast transport interfaces and ports on which **messages** can be received.

The unicast interfaces are used to receive messages from **DDSDataWriter** (p. 1589) entities in the domain.

9.32.3.20 multicast

`struct` **DDS_TransportMulticastQosPolicy** DDS_DataReaderQos::multicast

<<***extension***>> *(p. 432)* Multicast transport policy, **TRANSPORT_MULTICAST** (p. 645).

Specifies the multicast group addresses and ports on which **messages** can be received.

The multicast addresses are used to receive messages from **DDSDataWriter** (p. 1589) entities in the domain.

9.32.3.21 property

`struct` **DDS_PropertyQosPolicy** DDS_DataReaderQos::property

<<***basic***>> *(p. 201)* <<***extension***>> *(p. 432)* Property policy, **PROPERTY** (p. 617). See also `Property Reference Guide`.

9.32.3.22 service

`struct` **DDS_ServiceQosPolicy** DDS_DataReaderQos::service

<<***extension***>> *(p. 432)* Service policy, **SERVICE** (p. 636).

9.32.3.23 availability

`struct` **DDS_AvailabilityQosPolicy** DDS_DataReaderQos::availability

<<***extension***>> *(p. 432)* Availability policy, **AVAILABILITY** (p. 561).

9.32.3.24 subscription_name

`struct` **DDS_EntityNameQosPolicy** DDS_DataReaderQos::subscription_name

<<***extension***>> *(p. 432)* EntityName policy, **ENTITY_NAME** (p. 601).

9.32.3.25 transport_priority

```
struct DDS_TransportPriorityQosPolicy DDS_DataReaderQos::transport_priority
```

Transport priority policy, **TRANSPORT_PRIORITY** (p. 647).

9.32.3.26 type_support

```
struct DDS_TypeSupportQosPolicy DDS_DataReaderQos::type_support
```

<<***extension***>> *(p. 432)* type support data, **TYPESUPPORT** (p. 651).

Optional value that is passed to a type plugin's on_endpoint_attached and deserialization functions.

9.33 DDS_DataReaderResourceLimitsInstanceReplacementSettings Struct Reference

Instance replacement kind applied to each instance state.

Public Attributes

- **DDS_DataReaderInstanceRemovalKind alive_instance_removal**
  *Removal kind applied to alive (**DDS_ALIVE_INSTANCE_STATE** (p. 166)) instances.*
- **DDS_DataReaderInstanceRemovalKind disposed_instance_removal**
  *Removal kind applied to disposed (**DDS_NOT_ALIVE_DISPOSED_INSTANCE_STATE** (p. 166)) instances.*
- **DDS_DataReaderInstanceRemovalKind no_writers_instance_removal**
  *Removal kind applied to fully-unregistered (**DDS_NOT_ALIVE_NO_WRITERS_INSTANCE_STATE** (p. 166)) instances.*

9.33.1 Detailed Description

Instance replacement kind applied to each instance state.

[default]

- **DDS_DataReaderResourceLimitsInstanceReplacementSettings::alive_instance_removal** (p. 867) = **DDS_NO_INSTANCE_REMOVAL** (p. 566)
- **DDS_DataReaderResourceLimitsInstanceReplacementSettings::disposed_instance_removal** (p. 867) = **DDS_EMPTY_INSTANCE_REMOVAL** (p. 566)
- **DDS_DataReaderResourceLimitsInstanceReplacementSettings::no_writers_instance_removal** (p. 867) = **DDS_EMPTY_INSTANCE_REMOVAL** (p. 566)

See also

DDS_DataReaderResourceLimitsQosPolicy::instance_replacement (p. 883)

9.33.2 Member Data Documentation

9.33.2.1 alive_instance_removal

DDS_DataReaderInstanceRemovalKind DDS_DataReaderResourceLimitsInstanceReplacementSettings::alive_instance_removal

Removal kind applied to alive (**DDS_ALIVE_INSTANCE_STATE** (p. 166)) instances.

[default] DDS_NO_INSTANCE_REMOVAL (p. 566)

9.33.2.2 disposed_instance_removal

DDS_DataReaderInstanceRemovalKind DDS_DataReaderResourceLimitsInstanceReplacementSettings::disposed_instance_removal

Removal kind applied to disposed (**DDS_NOT_ALIVE_DISPOSED_INSTANCE_STATE** (p. 166)) instances.

[default] DDS_EMPTY_INSTANCE_REMOVAL (p. 566)

9.33.2.3 no_writers_instance_removal

DDS_DataReaderInstanceRemovalKind DDS_DataReaderResourceLimitsInstanceReplacementSettings::no_writers_instance_removal

Removal kind applied to fully-unregistered (**DDS_NOT_ALIVE_NO_WRITERS_INSTANCE_STATE** (p. 166)) instances.

[default] DDS_EMPTY_INSTANCE_REMOVAL (p. 566)

9.34 DDS_DataReaderResourceLimitsQosPolicy Struct Reference

<<***extension***>> *(p. 213)* Various settings that configure how a **DDSDataReader** (p. 1556) allocates and uses physical memory for internal resources.

Public Attributes

- **DDS_Long max_remote_writers**

  *<<**extension**>> (p. 213) The maximum number of remote writers from which a **DDSDataReader** (p. 1556) may read, including all instances.*
- **DDS_Long max_remote_writers_per_instance**

  *<<**extension**>> (p. 213) The maximum number of remote writers from which a **DDSDataReader** (p. 1556) may read a single instance.*
- **DDS_Long max_samples_per_remote_writer**

  *<<**extension**>> (p. 213) The maximum number of out-of-order samples from a given remote **DDSDataWriter** (p. 1589) that a **DDSDataReader** (p. 1556) may store when maintaining a reliable connection to the **DDSDataWriter** (p. 1589).*
- **DDS_Long max_infos**

  *<<**extension**>> (p. 213) The maximum number of info units that a **DDSDataReader** (p. 1556) can use to store **DDS_SampleInfo** (p. 1344).*
- **DDS_Long initial_remote_writers**

  *<<**extension**>> (p. 213) The initial number of remote writers from which a **DDSDataReader** (p. 1556) may read, including all instances.*
- **DDS_Long initial_remote_writers_per_instance**

  *<<**extension**>> (p. 213) The initial number of remote writers from which a **DDSDataReader** (p. 1556) may read a single instance.*
- **DDS_Long initial_infos**

  *<<**extension**>> (p. 213) The initial number of info units that a **DDSDataReader** (p. 1556) can have, which are used to store **DDS_SampleInfo** (p. 1344).*
- **DDS_Long initial_outstanding_reads**

  *<<**extension**>> (p. 213) The initial number of outstanding calls to read/take (or one of their variants) on the same **DDSDataReader** (p. 1556) for which memory has not been returned by calling **FooDataReader::return_loan** (p. 1940).*
- **DDS_Long max_outstanding_reads**

  *<<**extension**>> (p. 213) The maximum number of outstanding read/take calls (or one of their variants) on the same **DDSDataReader** (p. 1556) for which memory has not been returned by calling **FooDataReader::return_loan** (p. 1940).*
- **DDS_Long max_samples_per_read**

  *<<**extension**>> (p. 213) The maximum number of data samples that the application can receive from the middleware in a single call to **FooDataReader::read** (p. 1919) or **FooDataReader::take** (p. 1920). If more data exists in the middleware, the application will need to issue multiple read/take calls.*
- **DDS_Boolean disable_fragmentation_support**

  ***[DEPRECATED]** (p. **384**) <<**extension**>> (p. 213) Determines whether the **DDSDataReader** (p. 1556) can receive fragmented samples.*
- **DDS_Long max_fragmented_samples**

  *<<**extension**>> (p. 213) The maximum number of samples for which the **DDSDataReader** (p. 1556) may store fragments at a given point in time.*
- **DDS_Long initial_fragmented_samples**

  *<<**extension**>> (p. 213) The initial number of samples for which a **DDSDataReader** (p. 1556) may store fragments.*
- **DDS_Long max_fragmented_samples_per_remote_writer**

  *<<**extension**>> (p. 213) The maximum number of samples per remote writer for which a **DDSDataReader** (p. 1556) may store fragments.*
- **DDS_Long max_fragments_per_sample**

  *<<**extension**>> (p. 213) Maximum number of fragments for a single sample.*
- **DDS_Boolean dynamically_allocate_fragmented_samples**

  *<<**extension**>> (p. 213) Determines whether the **DDSDataReader** (p. 1556) pre-allocates storage for storing fragmented samples.*
- **DDS_Long max_total_instances**

*<<**extension**>> (p. 213) Maximum number of instances for which a DataReader will keep state.*

- **DDS_Long max_remote_virtual_writers**

  *<<**extension**>> (p. 213) The maximum number of remote virtual writers from which a **DDSDataReader** (p. 1556) may read, including all instances.*
- **DDS_Long initial_remote_virtual_writers**

  *<<**extension**>> (p. 213) The initial number of remote virtual writers from which a **DDSDataReader** (p. 1556) may read, including all instances.*
- **DDS_Long max_remote_virtual_writers_per_instance**

  *<<**extension**>> (p. 213) The maximum number of virtual remote writers that can be associated with an instance.*
- **DDS_Long initial_remote_virtual_writers_per_instance**

  *<<**extension**>> (p. 213) The initial number of virtual remote writers per instance.*
- **DDS_Long max_remote_writers_per_sample**

  *<<**extension**>> (p. 213) The maximum number of remote writers allowed to write the same sample.*
- **DDS_Long max_query_condition_filters**

  *<<**extension**>> (p. 213) The maximum number of query condition filters a reader is allowed.*
- **DDS_Long max_app_ack_response_length**

  *<<**extension**>> (p. 213) Maximum length of application-level acknowledgment response data*
- **DDS_Boolean keep_minimum_state_for_instances**

  *<<**extension**>> (p. 213) Whether or not keep a minimum instance state for up to **DDS_DataReaderResourceLimits↩ QosPolicy::max_total_instances** (p. 877).*
- **DDS_Long initial_topic_queries**

  *<<**extension**>> (p. 213) The initial number of TopicQueries allocated by a **DDSDataReader** (p. 1556).*
- **DDS_Long max_topic_queries**

  *<<**extension**>> (p. 213) The maximum number of active TopicQueries that a **DDSDataReader** (p. 1556) can create.*
- struct **DDS_AllocationSettings_t shmem_ref_transfer_mode_attached_segment_allocation**

  *<<**extension**>> (p. 213) Allocation resource for the shared memory segments attached by the **DDSDataReader** (p. 1556).*
- struct **DDS_DataReaderResourceLimitsInstanceReplacementSettings instance_replacement**

  *<<**extension**>> (p. 213) Sets the kind of instances allowed to be replaced for each instance state (**DDS_Instance↩ StateKind** (p. 165)) when a DataReader reaches **DDS_ResourceLimitsQosPolicy::max_instances** (p. 1311).*
- struct **DDS_Duration_t autopurge_remote_not_alive_writer_delay**

  *<<**extension**>> (p. 213) Maximum duration for which the **DDSDataReader** (p. 1556) will maintain information regarding a **DDSDataWriter** (p. 1589) once the **DDSDataWriter** (p. 1589) has become not alive.*
- **DDS_Boolean lazily_allocate_batch_resources**

  *<<**extension**>> (p. 213) Determines whether the **DDSDataReader** (p. 1556) pre-allocates resources for receiving batches of samples or lazily allocates them when needed.*

9.34.1 Detailed Description

*<<**extension**>> (p. 213)* Various settings that configure how a **DDSDataReader** (p. 1556) allocates and uses physical memory for internal resources.

DataReaders must allocate internal structures to handle the maximum number of DataWriters that may connect to it, whether or not a **DDSDataReader** (p. 1556) handles data fragmentation and how many data fragments that it may handle (for data samples larger than the MTU of the underlying network transport), how many simultaneous outstanding loans of internal memory holding data samples can be provided to user code, as well as others.

Most of these internal structures start at an initial size and, by default, will grow as needed by dynamically allocating additional memory. You may set fixed, maximum sizes for these internal structures if you want to bound the amount of memory that can be used by a **DDSDataReader** (p. 1556). By setting the initial size to the maximum size, you will prevent RTI Connext from dynamically allocating any memory after the creation of the **DDSDataReader** (p. 1556).

This QoS policy is an extension to the DDS standard.

Entity:

DDSDataReader (p. 1556)

Properties:

RxO (p. 556) = N/A
Changeable (p. 556) = **NO** (p. 556)

9.34.2 Member Data Documentation

9.34.2.1 max_remote_writers

DDS_Long DDS_DataReaderResourceLimitsQosPolicy::max_remote_writers

<<***extension***>> *(p. 213)* The maximum number of remote writers from which a **DDSDataReader** (p. 1556) may read, including all instances.

[default] DDS_LENGTH_UNLIMITED (p. 636)
[range] [1, 1 million] or **DDS_LENGTH_UNLIMITED** (p. 636), >= initial_remote_writers, >= max_remote_writers_↩ per_instance

For unkeyed types, this value has to be equal to max_remote_writers_per_instance if max_remote_writers_per_instance is not equal to **DDS_LENGTH_UNLIMITED** (p. 636).

Note: For efficiency, set max_remote_writers >= **DDS_DataReaderResourceLimitsQosPolicy::max_↩ remote_writers_per_instance** (p. 870).

Categories

Immutable (p. 333), **Middleware Memory Management** (p. 297)

9.34.2.2 max_remote_writers_per_instance

DDS_Long DDS_DataReaderResourceLimitsQosPolicy::max_remote_writers_per_instance

<<***extension***>> *(p. 213)* The maximum number of remote writers from which a **DDSDataReader** (p. 1556) may read a single instance.

[default] DDS_LENGTH_UNLIMITED (p. 636)

[range] [1, 1024] or **DDS_LENGTH_UNLIMITED** (p. 636), <= max_remote_writers or **DDS_LENGTH_UNLIMITED** (p. 636), >= initial_remote_writers_per_instance

For unkeyed types, this value has to be equal to max_remote_writers if it is not **DDS_LENGTH_UNLIMITED** (p. 636).

Note: For efficiency, set max_remote_writers_per_instance <= **DDS_DataReaderResourceLimitsQos↩ Policy::max_remote_writers** (p. 870)

Categories

Immutable (p. 333), **Middleware Memory Management** (p. 297)

9.34.2.3 max_samples_per_remote_writer

```
DDS_Long DDS_DataReaderResourceLimitsQosPolicy::max_samples_per_remote_writer
```

<<***extension***>> *(p. 213)* The maximum number of out-of-order samples from a given remote **DDSDataWriter** (p. 1589) that a **DDSDataReader** (p. 1556) may store when maintaining a reliable connection to the **DDSDataWriter** (p. 1589).

[default] DDS_LENGTH_UNLIMITED (p. 636)

[range] [1, 100 million] or **DDS_LENGTH_UNLIMITED** (p. 636), <= **DDS_ResourceLimitsQosPolicy::max_samples** (p. 1311)

Categories

Immutable (p. 333), **Middleware Memory Management** (p. 297)

9.34.2.4 max_infos

```
DDS_Long DDS_DataReaderResourceLimitsQosPolicy::max_infos
```

<<***extension***>> *(p. 213)* The maximum number of info units that a **DDSDataReader** (p. 1556) can use to store **DDS_SampleInfo** (p. 1344).

When read/take is called on a DataReader, the DataReader passes a sequence of data samples and an associated sample info sequence. The sample info sequence contains additional information for each data sample.

max_infos determines the resources allocated for storing sample info. This memory is loaned to the application when passing a sample info sequence.

Note that sample info is a snapshot, generated when read/take is called.

max_infos should not be less than max_samples.

[default] DDS_LENGTH_UNLIMITED (p. 636)

[range] [1, 1 million] or **DDS_LENGTH_UNLIMITED** (p. 636), >= initial_infos

Categories

Immutable (p. 333), **Middleware Memory Management** (p. 297)

9.34.2.5 initial_remote_writers

DDS_Long DDS_DataReaderResourceLimitsQosPolicy::initial_remote_writers

<<***extension***>> *(p. 213)* The initial number of remote writers from which a **DDSDataReader** (p. 1556) may read, including all instances.

[default] 2

[range] [1, 1 million], <= max_remote_writers

For unkeyed types this value has to be equal to initial_remote_writers_per_instance.

Note: For efficiency, set initial_remote_writers >= **DDS_DataReaderResourceLimitsQosPolicy::initial_remote_writers_per_instance** (p. 872).

Categories

Immutable (p. 333), **Middleware Memory Management** (p. 297)

9.34.2.6 initial_remote_writers_per_instance

DDS_Long DDS_DataReaderResourceLimitsQosPolicy::initial_remote_writers_per_instance

<<***extension***>> *(p. 213)* The initial number of remote writers from which a **DDSDataReader** (p. 1556) may read a single instance.

[default] 2

[range] [1,1024], <= max_remote_writers_per_instance

For unkeyed types this value has to be equal to initial_remote_writers.

Note: For efficiency, set initial_remote_writers_per_instance <= **DDS_DataReaderResourceLimitsQosPolicy::initial_remote_writers** (p. 871).

Categories

Immutable (p. 333), **Middleware Memory Management** (p. 297)

9.34.2.7 initial_infos

```
DDS_Long DDS_DataReaderResourceLimitsQosPolicy::initial_infos
```

<<***extension***>> *(p. 213)* The initial number of info units that a **DDSDataReader** (p. 1556) can have, which are used to store **DDS_SampleInfo** (p. 1344).

[default] 32

[range] [1,1 million], <= max_infos

Categories

Immutable (p. 333), **Middleware Memory Management** (p. 297)

9.34.2.8 initial_outstanding_reads

```
DDS_Long DDS_DataReaderResourceLimitsQosPolicy::initial_outstanding_reads
```

<<***extension***>> *(p. 213)* The initial number of outstanding calls to read/take (or one of their variants) on the same **DDSDataReader** (p. 1556) for which memory has not been returned by calling **FooDataReader::return_loan** (p. 1940).

[default] 2

[range] [1, 65536], <= max_outstanding_reads

Categories

Immutable (p. 333), **Middleware Memory Management** (p. 297)

9.34.2.9 max_outstanding_reads

```
DDS_Long DDS_DataReaderResourceLimitsQosPolicy::max_outstanding_reads
```

<<***extension***>> *(p. 213)* The maximum number of outstanding read/take calls (or one of their variants) on the same **DDSDataReader** (p. 1556) for which memory has not been returned by calling **FooDataReader::return_loan** (p. 1940).

[default] DDS_LENGTH_UNLIMITED (p. 636)

[range] [1, 65536] or **DDS_LENGTH_UNLIMITED** (p. 636), >= initial_outstanding_reads

Categories

Immutable (p. 333), **Middleware Memory Management** (p. 297)

9.34.2.10 max_samples_per_read

```
DDS_Long DDS_DataReaderResourceLimitsQosPolicy::max_samples_per_read
```

<<***extension***>> *(p. 213)* The maximum number of data samples that the application can receive from the middleware in a single call to **FooDataReader::read** (p. 1919) or **FooDataReader::take** (p. 1920). If more data exists in the middleware, the application will need to issue multiple read/take calls.

When reading data using listeners, the expected number of samples available for delivery in a single `take` call is typically small: usually just one, in the case of unbatched data, or the number of samples in a single batch, in the case of batched data. (See **DDS_BatchQosPolicy** (p. 810) for more information about this feature.) When polling for data or using a **DDSWaitSet** (p. 1897), however, multiple samples (or batches) could be retrieved at once, depending on the data rate.

A larger value for this parameter makes the API simpler to use at the expense of some additional memory consumption.

[default] 1024

[range] [1,65536]

Categories

Immutable (p. 333), **Middleware Memory Management** (p. 297)

9.34.2.11 disable_fragmentation_support

```
DDS_Boolean DDS_DataReaderResourceLimitsQosPolicy::disable_fragmentation_support
```

[DEPRECATED] (p. 384) <<***extension***>> *(p. 213)* Determines whether the **DDSDataReader** (p. 1556) can receive fragmented samples.

When fragmentation support is not needed, disabling fragmentation support will save some memory resources.

[default] DDS_BOOLEAN_FALSE (p. 513)

Categories

Immutable (p. 333), **Fragmentation** (p. 284), **Sending Large Data** (p. 290)

9.34.2.12 max_fragmented_samples

DDS_Long DDS_DataReaderResourceLimitsQosPolicy::max_fragmented_samples

<<***extension***>> *(p. 213)* The maximum number of samples for which the **DDSDataReader** (p. 1556) may store fragments at a given point in time.

At any given time, a **DDSDataReader** (p. 1556) may store fragments for up to `max_fragmented_samples` samples while waiting for the remaining fragments. These samples need not have consecutive sequence numbers and may have been sent by different **DDSDataWriter** (p. 1589) instances.

Once all fragments of a sample have been received, the sample is treated as a regular sample and becomes subject to standard QoS settings such as **DDS_ResourceLimitsQosPolicy::max_samples** (p. 1311).

The middleware will drop fragments if the `max_fragmented_samples` limit has been reached. For best-effort communication, the middleware will accept a fragment for a new sample, but drop the oldest fragmented sample from the same remote writer. For reliable communication, the middleware will drop fragments for any new samples until all fragments for at least one older sample from that writer have been received.

Only applies if **DDS_DataReaderResourceLimitsQosPolicy::disable_fragmentation_support** (p. 874) is **DDS_↩ BOOLEAN_FALSE** (p. 513).

[default] 1024

[range] [1, 1 million]

Categories

Immutable (p. 333), **Sample Memory Management** (p. 321), **Middleware Memory Management** (p. 297), **Fragmentation** (p. 284), **Sending Large Data** (p. 290)

9.34.2.13 initial_fragmented_samples

DDS_Long DDS_DataReaderResourceLimitsQosPolicy::initial_fragmented_samples

<<***extension***>> *(p. 213)* The initial number of samples for which a **DDSDataReader** (p. 1556) may store fragments.

Only applies if **DDS_DataReaderResourceLimitsQosPolicy::disable_fragmentation_support** (p. 874) is **DDS_↩ BOOLEAN_FALSE** (p. 513).

[default] 4

[range] [1,1024], <= max_fragmented_samples

Categories

Immutable (p. 333), **Sample Memory Management** (p. 321), **Middleware Memory Management** (p. 297), **Fragmentation** (p. 284), **Sending Large Data** (p. 290)

9.34.2.14 max_fragmented_samples_per_remote_writer

DDS_Long DDS_DataReaderResourceLimitsQosPolicy::max_fragmented_samples_per_remote_writer

<<***extension***>> *(p. 213)* The maximum number of samples per remote writer for which a **DDSDataReader** (p. 1556) may store fragments.

Logical limit so a single remote writer cannot consume all available resources.

Only applies if **DDS_DataReaderResourceLimitsQosPolicy::disable_fragmentation_support** (p. 874) is **DDS_↩ BOOLEAN_FALSE** (p. 513).

[default] 256

[range] [1, 1 million], <= max_fragmented_samples

Categories

Immutable (p. 333), **Sample Memory Management** (p. 321), **Middleware Memory Management** (p. 297), **Fragmentation** (p. 284), **Sending Large Data** (p. 290)

9.34.2.15 max_fragments_per_sample

DDS_Long DDS_DataReaderResourceLimitsQosPolicy::max_fragments_per_sample

<<***extension***>> *(p. 213)* Maximum number of fragments for a single sample.

Only applies if **DDS_DataReaderResourceLimitsQosPolicy::disable_fragmentation_support** (p. 874) is **DDS_↩ BOOLEAN_FALSE** (p. 513).

[default] DDS_LENGTH_UNLIMITED (p. 636)

[range] [1, 1 million] or **DDS_LENGTH_UNLIMITED** (p. 636)

Categories

Immutable (p. 333), **Sample Memory Management** (p. 321), **Middleware Memory Management** (p. 297), **Fragmentation** (p. 284), **Sending Large Data** (p. 290)

9.34.2.16 dynamically_allocate_fragmented_samples

```
DDS_Boolean DDS_DataReaderResourceLimitsQosPolicy::dynamically_allocate_fragmented_samples
```

<<***extension***>> *(p. 213)* Determines whether the **DDSDataReader** (p. 1556) pre-allocates storage for storing fragmented samples.

By default, the middleware does not allocate memory upfront, but instead allocates memory from the heap upon receiving the first fragment of a new sample. The amount of memory allocated equals the amount of memory needed to store all fragments in the sample. Once all fragments of a sample have been received, the sample is deserialized and stored in the regular receive queue. At that time, the dynamically allocated memory is freed again.

This QoS setting is useful for large, but variable-sized data types where upfront memory allocation for multiple samples based on the maximum possible sample size may be expensive. The main disadvantage of not pre-allocating memory is that one can no longer guarantee the middleware will have sufficient resources at runtime.

If `dynamically_allocate_fragmented_samples` is set to **DDS_BOOLEAN_FALSE** (p. 513), the middleware will allocate memory upfront for storing fragments for up to **DDS_DataReaderResourceLimitsQosPolicy::initial_fragmented_samples** (p. 875) samples. This memory may grow up to **DDS_DataReaderResourceLimitsQosPolicy::max_fragmented_samples** (p. 874) if needed.

Only applies if **DDS_DataReaderResourceLimitsQosPolicy::disable_fragmentation_support** (p. 874) is **DDS_BOOLEAN_FALSE** (p. 513).

[default] DDS_BOOLEAN_TRUE (p. 513)

Categories

Immutable (p. 333), **Sample Memory Management** (p. 321), **Middleware Memory Management** (p. 297), **Fragmentation** (p. 284), **Sending Large Data** (p. 290)

9.34.2.17 max_total_instances

```
DDS_Long DDS_DataReaderResourceLimitsQosPolicy::max_total_instances
```

<<***extension***>> *(p. 213)* Maximum number of instances for which a DataReader will keep state.

The maximum number of instances actively managed by a DataReader is determined by **DDS_ResourceLimitsQosPolicy::max_instances** (p. 1311).

These instances have associated DataWriters or samples in the DataReader's queue and are visible to the user through operations such as **FooDataReader::take** (p. 1920), **FooDataReader::read** (p. 1919), and **FooDataReader::get_key_value** (p. 1941).

The features Durable Reader State, MultiChannel DataWriters and RTI Persistence Service require RTI Connext to keep some internal state even for instances without DataWriters or samples in the DataReader's queue. The additional state is used to filter duplicate samples that could be coming from different DataWriter channels or from multiple executions of RTI Persistence Service.

The total maximum number of instances that will be managed by the middleware, including instances without associated DataWriters or samples, is determined by max_total_instances.

When a new instance is received, RTI Connext will check the resource limit **DDS_ResourceLimitsQosPolicy::max_instances** (p. 1311). If the limit is exceeded, RTI Connext will drop the sample with the reason LOST_BY_INSTANCES_LIMIT. If the limit is not exceeded, RTI Connext will check max_total_instances. If max_total_instances is exceeded, RTI Connext will replace an existing instance without DataWriters and samples with the new one. The application could receive duplicate samples for the replaced instance if it becomes alive again.

The max_total_instances limit is not used if **DDS_DataReaderResourceLimitsQosPolicy::keep_minimum_state_for_instances** (p. 881) is false, and in that case should be left at the default value.

[default] DDS_AUTO_MAX_TOTAL_INSTANCES (p. 567)

[range] [1, 1 million] or **DDS_LENGTH_UNLIMITED** (p. 636) or **DDS_AUTO_MAX_TOTAL_INSTANCES** (p. 567), >= **DDS_ResourceLimitsQosPolicy::max_instances** (p. 1311)

Categories

Immutable (p. 333), **Middleware Memory Management** (p. 297), **Instance Lifecycle** (p. 286)

9.34.2.18 max_remote_virtual_writers

```
DDS_Long DDS_DataReaderResourceLimitsQosPolicy::max_remote_virtual_writers
```

<<***extension***>> *(p. 213)* The maximum number of remote virtual writers from which a **DDSDataReader** (p. 1556) may read, including all instances.

When **DDS_PresentationQosPolicy::access_scope** (p. 1252) is set to **DDS_GROUP_PRESENTATION_QOS** (p. 616), this value determines the maximum number of DataWriter groups that can be managed by the **DDSSubscriber** (p. 1860) containing this **DDSDataReader** (p. 1556).

Since the **DDSSubscriber** (p. 1860) may contain more than one **DDSDataReader** (p. 1556), only the setting of the first applies.

[default] DDS_LENGTH_UNLIMITED (p. 636)
[range] [1, 1 million] or **DDS_LENGTH_UNLIMITED** (p. 636), >= initial_remote_virtual_writers, >= max_remote_virtual_writers_per_instance

Categories

Immutable (p. 333), **Middleware Memory Management** (p. 297)

9.34.2.19 initial_remote_virtual_writers

DDS_Long DDS_DataReaderResourceLimitsQosPolicy::initial_remote_virtual_writers

<<***extension***>> *(p. 213)* The initial number of remote virtual writers from which a **DDSDataReader** (p. 1556) may read, including all instances.

[default] 2
[range] [1, 1 million] or **DDS_LENGTH_UNLIMITED** (p. 636), <= max_remote_virtual_writers

Categories

Immutable (p. 333), **Middleware Memory Management** (p. 297)

9.34.2.20 max_remote_virtual_writers_per_instance

DDS_Long DDS_DataReaderResourceLimitsQosPolicy::max_remote_virtual_writers_per_instance

<<***extension***>> *(p. 213)* The maximum number of virtual remote writers that can be associated with an instance.

[default] DDS_LENGTH_UNLIMITED (p. 636)

[range] [1, 1024] or **DDS_LENGTH_UNLIMITED** (p. 636), >= initial_remote_virtual_writers_per_instance

For unkeyed types, this value is ignored.

The features of Durable Reader State and MultiChannel DataWriters, and RTI Persistence Service require RTI Connext to keep some internal state per virtual writer and instance that is used to filter duplicate samples. These duplicate samples could be coming from different DataWriter channels or from multiple executions of RTI Persistence Service.

Once an association between a remote virtual writer and an instance is established, it is permanent – it will not disappear even if the physical writer incarnating the virtual writer is destroyed.

If max_remote_virtual_writers_per_instance is exceeded for an instance, RTI Connext will not associate this instance with new virtual writers. Duplicates samples from these virtual writers will not be filtered on the reader.

If you are not using Durable Reader State, MultiChannel DataWriters or RTI Persistence Service in your system, you can set this property to 1 to optimize resources.

Categories

Immutable (p. 333), **Middleware Memory Management** (p. 297)

9.34.2.21 initial_remote_virtual_writers_per_instance

```
DDS_Long DDS_DataReaderResourceLimitsQosPolicy::initial_remote_virtual_writers_per_instance
```

<<***extension***>> *(p. 213)* The initial number of virtual remote writers per instance.

[default] 2

[range] [1, 1024], <= max_remote_virtual_writers_per_instance

For unkeyed types, this value is ignored.

Categories

Immutable (p. 333), **Middleware Memory Management** (p. 297)

9.34.2.22 max_remote_writers_per_sample

```
DDS_Long DDS_DataReaderResourceLimitsQosPolicy::max_remote_writers_per_sample
```

<<***extension***>> *(p. 213)* The maximum number of remote writers allowed to write the same sample.

One scenario in which two DataWriters may write the same sample is Persistence Service. The DataReader may receive the same sample coming from the original DataWriter and from a Persistence Service DataWriter. **[default]** 3

[range] [1, 1024]

Categories

Immutable (p. 333), **Middleware Memory Management** (p. 297)

9.34.2.23 max_query_condition_filters

DDS_Long DDS_DataReaderResourceLimitsQosPolicy::max_query_condition_filters

<<***extension***>> *(p. 213)* The maximum number of query condition filters a reader is allowed.

[default] 4

[range] [0, 32]

This value determines the maximum number of unique query condition content filters that a reader may create.

Each query condition content filter is comprised of both its `query_expression` and `query_parameters`. Two query conditions that have the same `query_expression` will require unique query condition filters if their `query_paramters` differ. Query conditions that differ only in their state masks will share the same query condition filter.

Categories

Immutable (p. 333), **Middleware Memory Management** (p. 297), **Content Filtering** (p. 260)

9.34.2.24 max_app_ack_response_length

DDS_Long DDS_DataReaderResourceLimitsQosPolicy::max_app_ack_response_length

<<***extension***>> *(p. 213)* Maximum length of application-level acknowledgment response data

The maximum length of response data in an application-level acknowledgment.

When set to zero, no response data is sent with application-level acknowledgments.

[default] 1

[range] [0, 32768]

Categories

Immutable (p. 333), **Middleware Memory Management** (p. 297), **Reliability** (p. 314), **Application Acknowledgement** (p. 253)

9.34.2.25 keep_minimum_state_for_instances

```
DDS_Boolean DDS_DataReaderResourceLimitsQosPolicy::keep_minimum_state_for_instances
```

<<***extension***>> *(p. 213)* Whether or not keep a minimum instance state for up to **DDS_DataReaderResourceLimitsQosPolicy::max_total_instances** (p. 877).

The features Durable Reader State, multi-channel DataWriters, and Persistence Service require RTI Connext to keep some minimal internal state even for instances without DataWriters or DDS samples in the DataReader's queue, or that have been purged due to a dispose. The additional state is used to filter duplicate DDS samples that could be coming from different DataWriter channels or from multiple executions of Persistence Service. The total maximum number of instances that will be managed by the middleware, including instances without associated DataWriters or DDS samples or that have been purged due to a dispose, is determined by **DDS_DataReaderResourceLimitsQosPolicy::max_total_instances** (p. 877).

This additional state will only be kept for up to max_total_instances if this field is set to **DDS_BOOLEAN_TRUE** (p. 513), otherwise the additional state will not be kept for any instances.

The minimum state includes information such as the source timestamp of the last sample received by the instance and the last sequence number received from a virtual GUID.

[default] DDS_BOOLEAN_TRUE (p. 513)

Categories

Immutable (p. 333), **Middleware Memory Management** (p. 297), **Instance Lifecycle** (p. 286)

9.34.2.26 initial_topic_queries

```
DDS_Long DDS_DataReaderResourceLimitsQosPolicy::initial_topic_queries
```

<<***extension***>> *(p. 213)* The initial number of TopicQueries allocated by a **DDSDataReader** (p. 1556).

[default] 1

See also

DDSTopicQuery (p. 1895)

Categories

Immutable (p. 333), **Middleware Memory Management** (p. 297), **Topic Queries** (p. 330)

9.34.2.27 max_topic_queries

DDS_Long DDS_DataReaderResourceLimitsQosPolicy::max_topic_queries

<<***extension***>> *(p. 213)* The maximum number of active TopicQueries that a **DDSDataReader** (p. 1556) can create.

Once this limit is reached, a **DDSDataReader** (p. 1556) can create more TopicQueries only if it deletes some of the previously created ones.

[default] DDS_LENGTH_UNLIMITED (p. 636)

See also

DDSTopicQuery (p. 1895)

Categories

Immutable (p. 333), **Middleware Memory Management** (p. 297), **Topic Queries** (p. 330)

9.34.2.28 shmem_ref_transfer_mode_attached_segment_allocation

struct **DDS_AllocationSettings_t** DDS_DataReaderResourceLimitsQosPolicy::shmem_ref_transfer_mode_↩attached_segment_allocation

<<***extension***>> *(p. 213)* Allocation resource for the shared memory segments attached by the **DDSDataReader** (p. 1556).

The max_count does not limit the total number of shared memory segments used by the **DDSDataReader** (p. 1556). When this limit is hit, the **DDSDataReader** (p. 1556) will try to detach from a segment that doesn't contain any loaned samples and attach to a new segment. If samples are loaned from all attached segments, then the **DDSDataReader** (p. 1556) will fail to attach to the new segment. This scenario will result in a sample loss.

[default] initial_count = **DDS_AUTO_COUNT** (p. 595) (**DDS_DataReaderResourceLimitsQosPolicy::initial_**↩**remote_writers** (p. 871)); max_count = **DDS_AUTO_COUNT** (p. 595) (**DDS_DataReaderResourceLimitsQos**↩**Policy::max_remote_writers** (p. 870)); incremental_count = **DDS_AUTO_COUNT** (p. 595) (0 if initial_count = max_↩count; -1 otherwise);

[range] See allowed ranges in struct **DDS_AllocationSettings_t** (p. 796)

Categories

Immutable (p. 333), **Middleware Memory Management** (p. 297), **Sending Large Data** (p. 290)

9.34.2.29 instance_replacement

`struct` **DDS_DataReaderResourceLimitsInstanceReplacementSettings** `DDS_DataReaderResourceLimitsQos↩ Policy::instance_replacement`

<<***extension***>> *(p. 213)* Sets the kind of instances allowed to be replaced for each instance state (**DDS_Instance↩ StateKind** (p. 165)) when a DataReader reaches **DDS_ResourceLimitsQosPolicy::max_instances** (p. 1311).

When **DDS_ResourceLimitsQosPolicy::max_instances** (p. 1311) is reached, a **DDSDataReader** (p. 1556) will try to make room for a new instance by attempting to reclaim an existing instance based on the instance replacement kinds specified by this field.

A DataReader can choose what kinds of instances can be replaced for each **DDS_InstanceStateKind** (p. 165) separately. This means, for example, that a DataReader can choose to not allow replacing alive (**DDS_ALIVE_INSTANCE↩ _STATE** (p. 166)) instances but allow replacement of empty disposed (**DDS_NOT_ALIVE_DISPOSED_INSTANCE_↩ STATE** (p. 166)) instances.

Only instances whose states match the specified kinds are eligible to be replaced. In addition, there must be no outstanding loans on any of the samples belonging to the instance for it to be considered for replacement.

For all kinds, a **DDSDataReader** (p. 1556) will replace the least-recently-updated instance satisfying that kind. An instance is considered 'updated' when a valid sample or dispose sample is received and accepted for that instance. When using **DDS_EXCLUSIVE_OWNERSHIP_QOS** (p. 613), only samples that are received from the owner of the instance will cause the instance to be considered updated. An instance is not considered updated when an unregister sample is received because the unregister message simply indicates that there is one less writer that has updates for the instance, not that the instance itself was updated.

If no replaceable instance exists, the sample for the new instance will be considered lost with lost reason **DDS_LOST↩ _BY_INSTANCES_LIMIT** (p. 140) and the instance will not be asserted into the DataReader queue.

[default]

- **DDS_DataReaderResourceLimitsInstanceReplacementSettings::alive_instance_removal** (p. 867) = **DDS↩ _NO_INSTANCE_REMOVAL** (p. 566)
- **DDS_DataReaderResourceLimitsInstanceReplacementSettings::disposed_instance_removal** (p. 867) = **DDS_EMPTY_INSTANCE_REMOVAL** (p. 566)
- **DDS_DataReaderResourceLimitsInstanceReplacementSettings::no_writers_instance_removal** (p. 867) = **DDS_EMPTY_INSTANCE_REMOVAL** (p. 566)

See also

DDS_DataReaderResourceLimitsInstanceReplacementSettings (p. 866)

Categories

Immutable (p. 333), **Middleware Memory Management** (p. 297), **Instance Lifecycle** (p. 286)

9.34.2.30 autopurge_remote_not_alive_writer_delay

`struct` **`DDS_Duration_t`** `DDS_DataReaderResourceLimitsQosPolicy::autopurge_remote_not_alive_writer_delay`

<<***extension***>> *(p. 213)* Maximum duration for which the **DDSDataReader** (p. 1556) will maintain information regarding a **DDSDataWriter** (p. 1589) once the **DDSDataWriter** (p. 1589) has become not alive.

After this time elapses, the **DDSDataReader** (p. 1556) will purge all internal information regarding the not alive **DDSDataWriter** (p. 1589).

See also

DDS_LivelinessQosPolicy (p. 1180) for more information on when a **DDSDataWriter** (p. 1589) is considered not alive.

When set to **DDS_DURATION_AUTO** (p. 523), this parameter is set to 10 times the value of **DDS_DiscoveryConfigQosPolicy::participant_liveliness_lease_duration** (p. 937).

This QoS only applies when the **DDSDataReader** (p. 1556) is using **DDS_RECOVER_INSTANCE_STATE_CONSISTENCY** (p. 634) for the **DDS_ReliabilityQosPolicy::instance_state_consistency_kind** (p. 1299) QoS. When using **DDS_RECOVER_INSTANCE_STATE_CONSISTENCY** (p. 634), a **DDSDataReader** (p. 1556) keeps state about all **DDSDataWriter** (p. 1589) entities and the instances they were writing in order to be able to transition those instances back to their correct state if liveliness with the **DDSDataWriter** (p. 1589) is recovered. This can cause unbounded memory growth if that state is never purged and **DDSDataWriter** (p. 1589) entities continuously come and go in a system. This QoS avoids that unbounded memory growth by setting a time at which that state will be purged.

This QoS should be set such that it is longer than the longest period of time for which a **DDSDataWriter** (p. 1589) and **DDSDataReader** (p. 1556) are expected to be disconnected and then reconnected in your system.

An alternative to using this QoS to purge the state is to set the **DDS_DataReaderResourceLimitsQosPolicy::max_remote_writers** (p. 870) QoS to a finite value. If that QoS is set to a finite value and the number of alive + not alive **DDSDataWriter** (p. 1589) entities reaches the limit when a new **DDSDataWriter** (p. 1589) is discovered, the oldest not alive **DDSDataWriter** (p. 1589) will be replaced.

[default] DDS_DURATION_AUTO (p. 523)

[range] $>$ 0 or **DDS_DURATION_AUTO** (p. 523)

Categories

Immutable (p. 333), **Middleware Memory Management** (p. 297), **Instance Lifecycle** (p. 286), **Liveliness** (p. 292)

9.34.2.31 lazily_allocate_batch_resources

```
DDS_Boolean DDS_DataReaderResourceLimitsQosPolicy::lazily_allocate_batch_resources
```

<<***extension***>> *(p. 213)* Determines whether the **DDSDataReader** (p. 1556) pre-allocates resources for receiving batches of samples or lazily allocates them when needed.

By default, the middleware does not allocate memory up front to manage the batches of samples received, but instead allocates memory from the heap upon receiving the first batch of samples.

Lazily allocating batching resources helps reduce memory consumption when batches are not used, but it may introduce some latency when receiving the first batch of samples, since memory needs to be allocated from the heap. If you know that your application will be using batches, you may set this value to **DDS_BOOLEAN_FALSE** (p. 513) if you are concerned about that latency.

[default] DDS_BOOLEAN_TRUE (p. 513)

Categories

Immutable (p. 333), **Middleware Memory Management** (p. 297)

9.35 DDS_DataRepresentationIdSeq Struct Reference

Declares IDL `sequence` < **DDS_DataRepresentationId_t** (p. 568) >

9.35.1 Detailed Description

Declares IDL `sequence` < **DDS_DataRepresentationId_t** (p. 568) >

See also

DDS_DataRepresentationId_t (p. 568)

9.36 DDS_DataRepresentationQosPolicy Struct Reference

<<***spec***>> *(p. 209)* This QoS policy contains a list of representation identifiers and compression settings used by **DDSDataWriter** (p. 1589) and **DDSDataReader** (p. 1556) entities to negotiate which data representation and compression settings to use.

Public Attributes

- struct **DDS_DataRepresentationIdSeq value**

  <<***spec***>> *(p. 209) Sequence of representation identifiers.*
- struct **DDS_CompressionSettings_t compression_settings**

  <<***basic***>> *(p. 201)* <<***extension***>> *(p. 213) Structure that contains the compression settings.*

9.36.1 Detailed Description

<<***spec***>> *(p. 209)* This QoS policy contains a list of representation identifiers and compression settings used by **DDSDataWriter** (p. 1589) and **DDSDataReader** (p. 1556) entities to negotiate which data representation and compression settings to use.

Entity:

DDSTopic (p. 1885), **DDSDataReader** (p. 1556), **DDSDataWriter** (p. 1589)

Status:

DDS_OFFERED_INCOMPATIBLE_QOS_STATUS (p. 540), **DDS_REQUESTED_INCOMPATIBLE_QOS_STATUS** (p. 540)

Properties:

RxO (p. 556) = YES
Changeable (p. 556) = **UNTIL ENABLE** (p. 556)

See also

DATA_REPRESENTATION (p. 567)

This policy has request-offer semantics for both representation and compression. For representation, a **DDSDataWriter** (p. 1589) may only offer a single representation. Attempting to put multiple representations in a **DDSDataWriter** (p. 1589) will result in **DDS_RETCODE_INCONSISTENT_POLICY** (p. 533). A **DDSDataWriter** (p. 1589) will use its offered policy to communicate with its matched **DDSDataReader** (p. 1556) entities. A **DDSDataReader** (p. 1556) requests one or more representations. If a **DDSDataWriter** (p. 1589) offers a representation that is contained within the sequence of the **DDSDataReader** (p. 1556), the offer satisfies the request and the policies are compatible. Otherwise, they are incompatible.

When representations are specified in the **DDS_TopicQos** (p. 1397), **DDSPublisher::copy_from_topic_qos** (p. 1834) copies the first element of the sequence, and **DDSSubscriber::copy_from_topic_qos** (p. 1876) copies the whole sequence.

For Compression, only the **DDS_CompressionSettings_t::compression_ids** (p. 826) is propagated and a **DDSDataWriter** (p. 1589) may only offer a single compression id. Attempting to put multiple compression_ids in a **DDSDataWriter** (p. 1589) will result in **DDS_RETCODE_INCONSISTENT_POLICY** (p. 533).

A **DDSDataWriter** (p. 1589) will use its offered compression algorithm to compress data that it sends to its matched **DDSDataReader** (p. 1556) entities. A **DDSDataReader** (p. 1556) requests zero or more compression algorithms. If a **DDSDataWriter** (p. 1589) offers a representation that is contained within the algorithms requested by the **DDSDataReader** (p. 1556), the offer satisfies the request and the policies are compatible. Otherwise, they are incompatible.

When compression IDs are specified in the **DDS_TopicQos** (p. 1397), **DDSPublisher::copy_from_topic_qos** (p. 1834) copies a single compression ID (see **DDS_CompressionSettings_t::compression_ids** (p. 826)), and **DDSSubscriber::copy_from_topic_qos** (p. 1876) copies all of the compression_ids.

9.36.2 Member Data Documentation

9.36.2.1 value

`struct` **`DDS_DataRepresentationIdSeq`** `DDS_DataRepresentationQosPolicy::value`

<<***spec***>> *(p. 209)* Sequence of representation identifiers.

[default] For **DDSTopic** (p. 1885), **DDSDataWriter** (p. 1589), and **DDSDataReader** (p. 1556), a list with one element: **DDS_AUTO_DATA_REPRESENTATION** (p. 569).

Note

An empty sequence is equivalent to a list with one element: **DDS_XCDR_DATA_REPRESENTATION** (p. 568).

Categories

Immutable (p. 333), **Matching** (p. 382), **Discovery Metadata** (p. 377), **Type System** (p. 331)

9.36.2.2 compression_settings

`struct` **`DDS_CompressionSettings_t`** `DDS_DataRepresentationQosPolicy::compression_settings`

<<***basic***>> *(p. 201)* <<***extension***>> *(p. 213)* Structure that contains the compression settings.

compression_ids:
[default] The value depends on the entity
DDSDataWriter (p. 1589) and **DDSTopic** (p. 1885) are set to **DDS_COMPRESSION_ID_MASK_NONE** (p. 571)
DDSDataReader (p. 1556) **DDS_COMPRESSION_ID_MASK_ALL** (p. 571)

writer_compression_level:
[default] The value depends on the entity
DDSDataWriter (p. 1589) and **DDSTopic** (p. 1885) are set to **DDS_COMPRESSION_LEVEL_DEFAULT** (p. 572)
DDSDataReader (p. 1556) NOT SUPPORTED

writer_compression_threshold:
[default] The value depends on the entity
DDSDataWriter (p. 1589) and **DDSTopic** (p. 1885) are set to **DDS_COMPRESSION_THRESHOLD_DEFAULT** (p. 572)
DDSDataReader (p. 1556) NOT SUPPORTED

See also

DDS_CompressionSettings_t (p. 826)

Categories

Immutable (p. 333), **Matching** (p. 382), **Discovery Metadata** (p. 377), **Sending Large Data** (p. 290)

9.37 DDS_DataTags Struct Reference

Definition of **DDS_DataTagQosPolicy** (p. 575).

Public Attributes

- struct **DDS_TagSeq tags**
  <<***spec***>> *(p. 209) Sequence of data tags.*

9.37.1 Detailed Description

Definition of **DDS_DataTagQosPolicy** (p. 575).

9.37.2 Member Data Documentation

9.37.2.1 tags

`struct` **`DDS_TagSeq`** `DDS_DataTags::tags`

<<***spec***>> *(p. 209)* Sequence of data tags.

[default] An empty list.

Categories

Immutable (p. 333), **Discovery Metadata** (p. 377), **Security** (p. 325)

9.38 DDS_DataWriterCacheStatus Struct Reference

<<***extension***>> *(p. 432)* The status of the DataWriter's cache. Provides information on cache related metrics such as the number of samples and instances in the DataWriter queue.

Public Attributes

- **DDS_LongLong sample_count_peak**

  *The highest value of **DDS_DataWriterCacheStatus::sample_count** (p. 890) over the lifetime of the DataWriter.*
- **DDS_LongLong sample_count**

  The number of samples in the DataWriter's queue. This statistic includes meta-samples that represent the unregistration or disposal of an instance.
- **DDS_LongLong alive_instance_count**

  *The number of instances currently in the DataWriter's queue that have an instance_state equal to **DDS_ALIVE_INSTANCE_STATE** (p. 166).*
- **DDS_LongLong alive_instance_count_peak**

  *The highest value of **DDS_DataWriterCacheStatus::alive_instance_count** (p. 891) over the lifetime of the DataWriter.*
- **DDS_LongLong disposed_instance_count**

  *The number of instances currently in the DataWriter's queue that have an instance_state equal to **DDS_NOT_ALIVE_DISPOSED_INSTANCE_STATE** (p. 166) (due to, for example, being disposed via the **FooDataWriter::dispose** (p. 1957) operation).*
- **DDS_LongLong disposed_instance_count_peak**

  *The highest value of **DDS_DataWriterCacheStatus::disposed_instance_count** (p. 891) over the lifetime of the DataWriter.*
- **DDS_LongLong unregistered_instance_count**

  *The number of instances currently in the DataWriter's queue that the DataWriter has unregistered from via the **FooDataWriter::unregister_instance** (p. 1948) operation.*
- **DDS_LongLong unregistered_instance_count_peak**

  *The highest value of **DDS_DataWriterCacheStatus::unregistered_instance_count** (p. 891) over the lifetime of the DataWriter.*

9.38.1 Detailed Description

<<***extension***>> *(p. 432)* The status of the DataWriter's cache. Provides information on cache related metrics such as the number of samples and instances in the DataWriter queue.

Entity:

DDSDataWriter (p. 1589)

9.38.2 Member Data Documentation

9.38.2.1 sample_count_peak

```
DDS_LongLong DDS_DataWriterCacheStatus::sample_count_peak
```

The highest value of **DDS_DataWriterCacheStatus::sample_count** (p. 890) over the lifetime of the DataWriter.

9.38.2.2 sample_count

```
DDS_LongLong DDS_DataWriterCacheStatus::sample_count
```

The number of samples in the DataWriter's queue. This statistic includes meta-samples that represent the unregistration or disposal of an instance.

9.38.2.3 alive_instance_count

```
DDS_LongLong DDS_DataWriterCacheStatus::alive_instance_count
```

The number of instances currently in the DataWriter's queue that have an instance_state equal to **DDS_ALIVE_↩ INSTANCE_STATE** (p. 166).

9.38.2.4 alive_instance_count_peak

```
DDS_LongLong DDS_DataWriterCacheStatus::alive_instance_count_peak
```

The highest value of **DDS_DataWriterCacheStatus::alive_instance_count** (p. 891) over the lifetime of the DataWriter.

9.38.2.5 disposed_instance_count

```
DDS_LongLong DDS_DataWriterCacheStatus::disposed_instance_count
```

The number of instances currently in the DataWriter's queue that have an instance_state equal to **DDS_NOT_↩ ALIVE_DISPOSED_INSTANCE_STATE** (p. 166) (due to, for example, being disposed via the **FooDataWriter::dispose** (p. 1957) operation).

9.38.2.6 disposed_instance_count_peak

```
DDS_LongLong DDS_DataWriterCacheStatus::disposed_instance_count_peak
```

The highest value of **DDS_DataWriterCacheStatus::disposed_instance_count** (p. 891) over the lifetime of the Data↩ Writer.

9.38.2.7 unregistered_instance_count

```
DDS_LongLong DDS_DataWriterCacheStatus::unregistered_instance_count
```

The number of instances currently in the DataWriter's queue that the DataWriter has unregistered from via the **Foo↩ DataWriter::unregister_instance** (p. 1948) operation.

9.38.2.8 unregistered_instance_count_peak

```
DDS_LongLong DDS_DataWriterCacheStatus::unregistered_instance_count_peak
```

The highest value of **DDS_DataWriterCacheStatus::unregistered_instance_count** (p. 891) over the lifetime of the DataWriter.

9.39 DDS_DataWriterProtocolQosPolicy Struct Reference

<<***extension***>> *(p. 213)* Protocol that applies only to **DDSDataWriter** (p. 1589) instances.

Public Attributes

- struct **DDS_GUID_t virtual_guid**

  <<***extension***>> *(p. 213) The virtual GUID (Global Unique Identifier).*
- **DDS_UnsignedLong rtps_object_id**

  <<***extension***>> *(p. 213) The RTPS Object ID.*
- **DDS_Boolean push_on_write**

  <<***extension***>> *(p. 213) Whether to push sample out when write is called.*
- **DDS_Boolean disable_positive_acks**

  <<***extension***>> *(p. 213) Controls whether or not the writer expects positive acknowledgements from matching readers.*
- **DDS_Boolean disable_inline_keyhash**

  <<***extension***>> *(p. 213) Controls whether or not a keyhash is propagated on the wire with each sample.*
- **DDS_Boolean serialize_key_with_dispose**

  <<***extension***>> *(p. 213) Controls whether or not the serialized key is propagated on the wire with dispose samples.*
- **DDS_Boolean propagate_app_ack_with_no_response**

  <<***extension***>> *(p. 213) Controls whether or not a **DDSDataWriter** (p. 1589) receives **DDSDataWriterListener::on↩ _application_acknowledgment** (p. 1617) notifications with an empty or invalid response.*
- struct **DDS_RtpsReliableWriterProtocol_t rtps_reliable_writer**

  <<***extension***>> *(p. 213) RTPS protocol-related configuration settings for the RTPS reliable writer associated with a **DDSDataWriter** (p. 1589). This parameter only has effect if both the writer and the matching reader are configured with **DDS_RELIABLE_RELIABILITY_QOS** (p. 633) **DDS_ReliabilityQosPolicyKind** (p. 632).*
- struct **DDS_SequenceNumber_t initial_virtual_sequence_number**

  <<***extension***>> *(p. 213) Determines, the initial virtual sequence number for this DataWriter.*

9.39.1 Detailed Description

<<***extension***>> *(p. 213)* Protocol that applies only to **DDSDataWriter** (p. 1589) instances.

DDS has a standard protocol for packet (user and meta data) exchange between applications using DDS for communications. This QoS policy and **DDS_DataWriterProtocolQosPolicy** (p. 892) give you control over configurable portions of the protocol, including the configuration of the reliable data delivery mechanism of the protocol on a per DataWriter or DataReader basis.

These configuration parameters control timing, timeouts, and give you the ability to tradeoff between speed of data loss detection and repair versus network and CPU bandwidth used to maintain reliability.

It is important to tune the reliability protocol (on a per **DDSDataWriter** (p. 1589) and **DDSDataReader** (p. 1556) basis) to meet the requirements of the end-user application so that data can be sent between DataWriters and DataReaders in an efficient and optimal manner in the presence of data loss.

You can also use this QoS policy to control how RTI Connext responds to "slow" reliable DataReaders or ones that disconnect or are otherwise lost. See **DDS_ReliabilityQosPolicy** (p. 1295) for more information on the per-DataReader/DataWriter reliability configuration. **DDS_HistoryQosPolicy** (p. 1162) and **DDS_ResourceLimitsQosPolicy** (p. 1308) also play an important role in the DDS reliable protocol.

This QoS policy is an extension to the DDS standard.

Entity:

DDSDataWriter (p. 1589)

Properties:

RxO (p. 556) = N/A
Changeable (p. 556) = **NO** (p. 556)

9.39.2 Member Data Documentation

9.39.2.1 virtual_guid

`struct DDS_GUID_t DDS_DataWriterProtocolQosPolicy::virtual_guid`

<<***extension***>> *(p. 213)* The virtual GUID (Global Unique Identifier).

The virtual GUID is used to uniquely identify different incarnations of the same **DDSDataWriter** (p. 1589).

RTI Connext uses the virtual GUID to associate a persisted writer history to a specific **DDSDataWriter** (p. 1589).

The RTI Connext Persistence Service uses the virtual GUID to send samples on behalf of the original **DDSDataWriter** (p. 1589).

[default] DDS_GUID_AUTO (p. 527)

Categories

Immutable (p. 333), **Discovery Metadata** (p. 377), **Data Availability** (p. 261)

9.39.2.2 rtps_object_id

```
DDS_UnsignedLong DDS_DataWriterProtocolQosPolicy::rtps_object_id
```

<<***extension***>> *(p. 213)* The RTPS Object ID.

This value is used to determine the RTPS object ID of a data writer according to the DDS-RTPS Interoperability Wire Protocol.

Only the last 3 bytes are used; the most significant byte is ignored.

If the default value is specified, RTI Connext will automatically assign the object ID based on a counter value (per participant) starting at 0x00800000. That value is incremented for each new data writer.

A rtps_object_id value in the interval [0x00800000,0x00ffffff] may collide with the automatic values assigned by RTI Connext. In those cases, the recomendation is not to use automatic object ID assigment.

[default] DDS_RTPS_AUTO_ID (p. **??**)

[range] [0,0x00ffffff]

Categories

Immutable (p. 333)

9.39.2.3 push_on_write

```
DDS_Boolean DDS_DataWriterProtocolQosPolicy::push_on_write
```

<<***extension***>> *(p. 213)* Whether to push sample out when write is called.

If set to **DDS_BOOLEAN_TRUE** (p. 513) (the default), the writer will send a sample every time write is called. Otherwise, the sample is put into the queue waiting for a NACK from remote reader(s) to be sent out.

[default] DDS_BOOLEAN_TRUE (p. 513)

Categories

Immutable (p. 333), **Reliability** (p. 314), **Data Availability** (p. 261)

9.39.2.4 disable_positive_acks

```
DDS_Boolean DDS_DataWriterProtocolQosPolicy::disable_positive_acks
```

<<***extension***>> *(p. 213)* Controls whether or not the writer expects positive acknowledgements from matching readers.

If set to **DDS_BOOLEAN_TRUE** (p. 513), the writer does not expect readers to send send positive acknowledgments to the writer. Consequently, instead of keeping a sample queued until all readers have positively acknowledged it, the writer will keep a sample for at least **DDS_RtpsReliableWriterProtocol_t::disable_positive_acks_min_sample_keep_duration** (p. 1330), after which the sample is logically considered as positively acknowledged.

If set to **DDS_BOOLEAN_FALSE** (p. 513) (the default), the writer expects to receive positive acknowledgements from its acknowledging readers (**DDS_DataReaderProtocolQosPolicy::disable_positive_acks** (p. 845) = **DDS_BOOLEAN_FALSE** (p. 513)) and it applies the keep-duration to its non-acknowledging readers (**DDS_DataReaderProtocolQosPolicy::disable_positive_acks** (p. 845) = **DDS_BOOLEAN_TRUE** (p. 513)).

A writer with both acknowledging and non-acknowledging readers keeps a sample queued until acknowledgements have been received from all acknowledging readers and the keep-duration has elapsed for non-acknowledging readers.

[default] DDS_BOOLEAN_FALSE (p. 513)

Categories

Immutable (p. 333), **Discovery Metadata** (p. 377), **Reliability** (p. 314)

9.39.2.5 disable_inline_keyhash

```
DDS_Boolean DDS_DataWriterProtocolQosPolicy::disable_inline_keyhash
```

<<***extension***>> *(p. 213)* Controls whether or not a keyhash is propagated on the wire with each sample.

This field only applies to keyed writers.

With each key, RTI Connext associates an internal 16-byte representation, called a keyhash.

When this field is **DDS_BOOLEAN_FALSE** (p. 513), the keyhash is sent on the wire with every data instance.

When this field is **DDS_BOOLEAN_TRUE** (p. 513), the keyhash is not sent on the wire and the readers must compute the value using the received data.

If the *reader* is CPU bound, sending the keyhash on the wire may increase performance, because the reader does not have to get the keyhash from the data.

If the *writer* is CPU bound, sending the keyhash on the wire may decrease performance, because it requires more bandwidth (16 more bytes per sample).

Note: Setting disable_inline_keyhash to **DDS_BOOLEAN_TRUE** (p. 513) is not compatible with using RTI Real-Time Connect or RTI Recorder.

[default] DDS_BOOLEAN_FALSE (p. 513)

Categories

Immutable (p. 333), **Instance Lifecycle** (p. 286)

9.39.2.6 serialize_key_with_dispose

```
DDS_Boolean DDS_DataWriterProtocolQosPolicy::serialize_key_with_dispose
```

<<***extension***>> *(p. 213)* Controls whether or not the serialized key is propagated on the wire with dispose samples.

This field only applies to keyed writers.

We recommend setting this field to **DDS_BOOLEAN_TRUE** (p. 513) if there are DataReaders where **DDS_Data↩ ReaderProtocolQosPolicy::propagate_dispose_of_unregistered_instances** (p. 845) is also **DDS_BOOLEAN_↩ TRUE** (p. 513).

When setting serialize_key_with_dispose to FALSE, only a key hash is included in the dispose meta-sample sent by a DataWriter for a dispose action. If a dispose meta-sample only includes the key hash, then DataReaders must have previously received an actual data sample for the instance being disposed, in order for a DataReader to map a key hash/instance handle to actual key values.

If an actual data sample was never received for an instance and serialize_key_with_dispose is set to FALSE, then the DataReader application will not be able to determine the value of the key that was disposed, since **FooDataReader↩ ::get_key_value** (p. 1941) will not be able to map an instance handle to actual key values.

By setting serialize_key_with_dispose to TRUE, the values of the key members of a data type will be sent in the dispose meta-sample for a dispose action by the DataWriter. This allows the DataReader to map an instance handle to the values of the key members even when receiving a dispose meta-sample without previously having received a data sample for the instance.

Important: When this field is **DDS_BOOLEAN_TRUE** (p. 513), batching will not be compatible with RTI Connext 4.3e, 4.4b, or 4.4c. The **DDSDataReader** (p. 1556) entities will receive incorrect data and/or encounter deserialization errors.

[default] DDS_BOOLEAN_FALSE (p. 513)

Categories

Immutable (p. 333), **Instance Lifecycle** (p. 286)

9.39.2.7 propagate_app_ack_with_no_response

```
DDS_Boolean DDS_DataWriterProtocolQosPolicy::propagate_app_ack_with_no_response
```

<<***extension***>> *(p. 213)* Controls whether or not a **DDSDataWriter** (p. 1589) receives **DDSDataWriterListener↩ ::on_application_acknowledgment** (p. 1617) notifications with an empty or invalid response.

When this field is set to **DDS_BOOLEAN_FALSE** (p. 513), the callback **DDSDataWriterListener::on_application_↩ acknowledgment** (p. 1617) will not be invoked if the sample being acknowledged has an empty or invalid response.

[default] DDS_BOOLEAN_TRUE (p. 513)

Categories

Immutable (p. 333), **Reliability** (p. 314), **Application Acknowledgement** (p. 253), **Durability** (p. 278)

9.39.2.8 rtps_reliable_writer

```
struct DDS_RtpsReliableWriterProtocol_t DDS_DataWriterProtocolQosPolicy::rtps_reliable_writer
```

<<***extension***>> *(p. 213)* RTPS protocol-related configuration settings for the RTPS reliable writer associated with a **DDSDataWriter** (p. 1589). This parameter only has effect if both the writer and the matching reader are configured with **DDS_RELIABLE_RELIABILITY_QOS** (p. 633) **DDS_ReliabilityQosPolicyKind** (p. 632).

For details, refer to the **DDS_RtpsReliableWriterProtocol_t** (p. 1320)

[default] [default] See **DDS_RtpsReliableWriterProtocol_t** (p. 1320)

Categories

Immutable (p. 333), **Reliability** (p. 314)

9.39.2.9 initial_virtual_sequence_number

```
struct DDS_SequenceNumber_t DDS_DataWriterProtocolQosPolicy::initial_virtual_sequence_number
```

<<***extension***>> *(p. 213)* Determines, the initial virtual sequence number for this DataWriter.

By default, the virtual sequence number of the first sample published by a DataWriter will be 1 for DataWriters that do not use durable writer history. For durable writers, the default virtual sequence number will be the last sequence number they published in a previous execution, plus one. So, when a non-durable DataWriter is restarted and must continue communicating with the same DataReaders, its samples start over with sequence number 1. Durable DataWriters start over where the last sequence number left off, plus one.

This QoS setting allows overwriting the default initial virtual sequence number.

Normally, this parameter is not expected to be modified; however, in some scenarios when continuing communication after restarting, applications may require the DataWriter's virtual sequence number to start at something other than the value described above. An example would be to enable non-durable DataWriters to start at the last sequence number published, plus one, similar to the durable DataWriter. This property enables you to make such a configuration, if desired.

The virtual sequence number can be overwritten as well on a per sample basis by updating **DDS_WriteParams_t**↩ **::identity** (p. 1520) in the **FooDataWriter::write_w_params** (p. 1956).

[default] DDS_AUTO_SEQUENCE_NUMBER (p. 529)

Categories

Immutable (p. 333)

9.40 DDS_DataWriterProtocolStatus Struct Reference

<<***extension***>> *(p. 432)* The status of a writer's internal protocol related metrics, like the number of samples pushed, pulled, filtered; and status of wire protocol traffic.

Public Attributes

- **DDS_LongLong pushed_sample_count**

  The number of user samples pushed on write from a local DataWriter to a matching remote DataReader.
- **DDS_LongLong pushed_sample_count_change**

  *The change in **DDS_DataWriterProtocolStatus::pushed_sample_count** (p. 900) since the last time the status was read.*
- **DDS_LongLong pushed_sample_bytes**

  The number of bytes of user samples pushed on write from a local DataWriter to a matching remote DataReader.
- **DDS_LongLong pushed_sample_bytes_change**

  *The change in **DDS_DataWriterProtocolStatus::pushed_sample_bytes** (p. 900) since the last time the status was read.*
- **DDS_LongLong filtered_sample_count**

  ***[Not supported.]** The number of user samples preemptively filtered by a local DataWriter due to Content-Filtered Topics.*
- **DDS_LongLong filtered_sample_count_change**

  ***[Not supported.]** The incremental change in the number of user samples preemptively filtered by a local DataWriter due to Content-Filtered Topics since the last time the status was read.*
- **DDS_LongLong filtered_sample_bytes**

  ***[Not supported.]** The number of bytes of user samples preemptively filtered by a local DataWriter due to Content-Filtered Topics.*
- **DDS_LongLong filtered_sample_bytes_change**

  ***[Not supported.]** The incremental change in the number of bytes of user samples preemptively filtered by a local Data↩ Writer due to Content-Filtered Topics since the last time the status was read.*
- **DDS_LongLong sent_heartbeat_count**

  The number of Heartbeats sent between a local DataWriter and matching remote DataReader.
- **DDS_LongLong sent_heartbeat_count_change**

  *The change in **DDS_DataWriterProtocolStatus::sent_heartbeat_count** (p. 901) since the last time the status was read.*
- **DDS_LongLong sent_heartbeat_bytes**

  The number of bytes of Heartbeats sent between a local DataWriter and matching remote DataReader.
- **DDS_LongLong sent_heartbeat_bytes_change**

  *The change in **DDS_DataWriterProtocolStatus::sent_heartbeat_bytes** (p. 902) since the last time the status was read.*
- **DDS_LongLong pulled_sample_count**

  The number of user samples pulled from local DataWriter by matching DataReaders.
- **DDS_LongLong pulled_sample_count_change**

  *The change in **DDS_DataWriterProtocolStatus::pulled_sample_count** (p. 902) since the last time the status was read.*
- **DDS_LongLong pulled_sample_bytes**

  The number of bytes of user samples pulled from local DataWriter by matching DataReaders.
- **DDS_LongLong pulled_sample_bytes_change**

  *The change in **DDS_DataWriterProtocolStatus::pulled_sample_bytes** (p. 903) since the last time the status was read.*
- **DDS_LongLong received_ack_count**

  The number of ACKs from a remote DataReader received by a local DataWriter.
- **DDS_LongLong received_ack_count_change**

  *The change in **DDS_DataWriterProtocolStatus::received_ack_count** (p. 903) since the last time the status was read.*
- **DDS_LongLong received_ack_bytes**

  The number of bytes of ACKs from a remote DataReader received by a local DataWriter.
- **DDS_LongLong received_ack_bytes_change**

  *The change in **DDS_DataWriterProtocolStatus::received_ack_bytes** (p. 903) since the last time the status was read.*
- **DDS_LongLong received_nack_count**

  The number of NACKs from a remote DataReader received by a local DataWriter.
- **DDS_LongLong received_nack_count_change**

*The change in **DDS_DataWriterProtocolStatus::received_nack_count** (p. 904) since the last time the status was read.*

- **DDS_LongLong received_nack_bytes**

  The number of bytes of NACKs from a remote DataReader received by a local DataWriter.
- **DDS_LongLong received_nack_bytes_change**

  *The change in **DDS_DataWriterProtocolStatus::received_nack_bytes** (p. 904) since the last time the status was read.*
- **DDS_LongLong sent_gap_count**

  The number of GAPs sent from local DataWriter to matching remote DataReaders.
- **DDS_LongLong sent_gap_count_change**

  *The change in **DDS_DataWriterProtocolStatus::sent_gap_count** (p. 904) since the last time the status was read.*
- **DDS_LongLong sent_gap_bytes**

  The number of bytes of GAPs sent from local DataWriter to matching remote DataReaders.
- **DDS_LongLong sent_gap_bytes_change**

  *The change in **DDS_DataWriterProtocolStatus::sent_gap_bytes** (p. 905) since the last time the status was read.*
- **DDS_LongLong rejected_sample_count**

  [Not supported.]
- **DDS_LongLong rejected_sample_count_change**

  [Not supported.]
- **DDS_Long send_window_size**

  Current maximum number of outstanding samples allowed in the DataWriter's queue.
- struct **DDS_SequenceNumber_t first_available_sample_sequence_number**

  The sequence number of the first available sample currently queued in the local DataWriter.
- struct **DDS_SequenceNumber_t last_available_sample_sequence_number**

  The sequence number of the last available sample currently queued in the local DataWriter.
- struct **DDS_SequenceNumber_t first_unacknowledged_sample_sequence_number**

  The sequence number of the first unacknowledged sample currently queued in the local DataWriter.
- struct **DDS_SequenceNumber_t first_available_sample_virtual_sequence_number**

  The virtual sequence number of the first available sample currently queued in the local DataWriter.
- struct **DDS_SequenceNumber_t last_available_sample_virtual_sequence_number**

  The virtual sequence number of the last available sample currently queued in the local DataWriter.
- struct **DDS_SequenceNumber_t first_unacknowledged_sample_virtual_sequence_number**

  The virtual sequence number of the first unacknowledged sample currently queued in the local DataWriter.
- **DDS_InstanceHandle_t first_unacknowledged_sample_subscription_handle**

  The handle of a remote DataReader that has not acknowledged the first unacknowledged sample of the local DataWriter.
- struct **DDS_SequenceNumber_t first_unelapsed_keep_duration_sample_sequence_number**

  The sequence number of the first sample whose keep duration has not yet elapsed.
- **DDS_LongLong pushed_fragment_count**

  The number of DATA_FRAG messages that have been pushed by this DataWriter.
- **DDS_LongLong pushed_fragment_bytes**

  The number of bytes of DATA_FRAG messages that have been pushed by this DataWriter.
- **DDS_LongLong pulled_fragment_count**

  The number of DATA_FRAG messages that have been pulled from this DataWriter.
- **DDS_LongLong pulled_fragment_bytes**

  The number of bytes of DATA_FRAG messages that have been pulled from this DataWriter.
- **DDS_LongLong received_nack_fragment_count**

  The number of NACK_FRAG messages that have been received by this DataWriter.
- **DDS_LongLong received_nack_fragment_bytes**

  The number of bytes of NACK_FRAG messages that have been received by this DataWriter.

9.40.1 Detailed Description

<<***extension***>> *(p. 432)* The status of a writer's internal protocol related metrics, like the number of samples pushed, pulled, filtered; and status of wire protocol traffic.

Entity:

DDSDataWriter (p. 1589)

9.40.2 Member Data Documentation

9.40.2.1 pushed_sample_count

DDS_LongLong DDS_DataWriterProtocolStatus::pushed_sample_count

The number of user samples pushed on write from a local DataWriter to a matching remote DataReader.

Counts protocol (RTPS) messages pushed by a DataWriter when writing, unregistering, and disposing. The count is the number of sends done internally, and it may be greater than the number of user writes.

For large data, counts whole samples, not fragments. The fragment count is tracked in the **DDS_DataWriterProtocol↩ Status::pushed_fragment_count** (p. 907) statistic.

9.40.2.2 pushed_sample_count_change

DDS_LongLong DDS_DataWriterProtocolStatus::pushed_sample_count_change

The change in **DDS_DataWriterProtocolStatus::pushed_sample_count** (p. 900) since the last time the status was read.

Counts protocol (RTPS) messages pushed by a DataWriter when writing, unregistering, and disposing.

For large data, counts whole samples, not fragments.

9.40.2.3 pushed_sample_bytes

DDS_LongLong DDS_DataWriterProtocolStatus::pushed_sample_bytes

The number of bytes of user samples pushed on write from a local DataWriter to a matching remote DataReader.

Counts bytes of protocol (RTPS) messages pushed by a DataWriter when writing, unregistering, and disposing. The count of bytes corresponds to the number of sends done internally, and it may be greater than the number of user writes.

When data fragmentation is used, this statistic is incremented as fragments are written.

9.40.2.4 pushed_sample_bytes_change

DDS_LongLong DDS_DataWriterProtocolStatus::pushed_sample_bytes_change

The change in **DDS_DataWriterProtocolStatus::pushed_sample_bytes** (p. 900) since the last time the status was read.

Counts bytes of protocol (RTPS) messages pushed by a DataWriter when writing, unregistering, and disposing.

When data fragmentation is used, this statistic is incremented as fragments are written.

9.40.2.5 filtered_sample_count

DDS_LongLong DDS_DataWriterProtocolStatus::filtered_sample_count

[Not supported.] The number of user samples preemptively filtered by a local DataWriter due to Content-Filtered Topics.

9.40.2.6 filtered_sample_count_change

DDS_LongLong DDS_DataWriterProtocolStatus::filtered_sample_count_change

[Not supported.] The incremental change in the number of user samples preemptively filtered by a local DataWriter due to Content-Filtered Topics since the last time the status was read.

9.40.2.7 filtered_sample_bytes

DDS_LongLong DDS_DataWriterProtocolStatus::filtered_sample_bytes

[Not supported.] The number of bytes of user samples preemptively filtered by a local DataWriter due to Content-↩ Filtered Topics.

9.40.2.8 filtered_sample_bytes_change

DDS_LongLong DDS_DataWriterProtocolStatus::filtered_sample_bytes_change

[Not supported.] The incremental change in the number of bytes of user samples preemptively filtered by a local DataWriter due to Content-Filtered Topics since the last time the status was read.

9.40.2.9 sent_heartbeat_count

DDS_LongLong DDS_DataWriterProtocolStatus::sent_heartbeat_count

The number of Heartbeats sent between a local DataWriter and matching remote DataReader.

Because periodic and piggyback heartbeats are sent to remote readers and their locators differently in different situations, when a reader has more than one locator, this count may be larger than expected, to reflect the sending of Heartbeats to the multiple locators.

9.40.2.10 sent_heartbeat_count_change

DDS_LongLong DDS_DataWriterProtocolStatus::sent_heartbeat_count_change

The change in **DDS_DataWriterProtocolStatus::sent_heartbeat_count** (p. 901) since the last time the status was read.

9.40.2.11 sent_heartbeat_bytes

DDS_LongLong DDS_DataWriterProtocolStatus::sent_heartbeat_bytes

The number of bytes of Heartbeats sent between a local DataWriter and matching remote DataReader.

Because periodic and piggyback heartbeats are sent to remote readers and their locators differently in different situations, when a reader has more than one locator, this count may be larger than expected, to reflect the sending of Heartbeats to the multiple locators.

9.40.2.12 sent_heartbeat_bytes_change

DDS_LongLong DDS_DataWriterProtocolStatus::sent_heartbeat_bytes_change

The change in **DDS_DataWriterProtocolStatus::sent_heartbeat_bytes** (p. 902) since the last time the status was read.

9.40.2.13 pulled_sample_count

DDS_LongLong DDS_DataWriterProtocolStatus::pulled_sample_count

The number of user samples pulled from local DataWriter by matching DataReaders.

Pulled samples are samples sent for repairs, for late joiners, and all samples sent by the local DataWriter when **DDS_DataWriterProtocolQosPolicy::push_on_write** (p. 894) is **DDS_BOOLEAN_FALSE** (p. 513).

When data fragmentation is used, this statistic is incremented as fragments are written.

9.40.2.14 pulled_sample_count_change

```
DDS_LongLong DDS_DataWriterProtocolStatus::pulled_sample_count_change
```

The change in **DDS_DataWriterProtocolStatus::pulled_sample_count** (p. 902) since the last time the status was read.

Pulled samples are samples sent for repairs, for late joiners, and all samples sent by the local DataWriter when **DDS_DataWriterProtocolQosPolicy::push_on_write** (p. 894) is **DDS_BOOLEAN_FALSE** (p. 513).

For large data, counts whole samples, not fragments.

9.40.2.15 pulled_sample_bytes

```
DDS_LongLong DDS_DataWriterProtocolStatus::pulled_sample_bytes
```

The number of bytes of user samples pulled from local DataWriter by matching DataReaders.

Pulled samples are samples sent for repairs, for late joiners, and all samples sent by the local DataWriter when **DDS_DataWriterProtocolQosPolicy::push_on_write** (p. 894) is **DDS_BOOLEAN_FALSE** (p. 513).

When data fragmentation is used, this statistic is incremented as fragments are written.

9.40.2.16 pulled_sample_bytes_change

```
DDS_LongLong DDS_DataWriterProtocolStatus::pulled_sample_bytes_change
```

The change in **DDS_DataWriterProtocolStatus::pulled_sample_bytes** (p. 903) since the last time the status was read.

Pulled samples are samples sent for repairs, for late joiners, and all samples sent by the local DataWriter when **DDS_DataWriterProtocolQosPolicy::push_on_write** (p. 894) is **DDS_BOOLEAN_FALSE** (p. 513).

For large data, counts bytes of whole samples, not fragments.

9.40.2.17 received_ack_count

```
DDS_LongLong DDS_DataWriterProtocolStatus::received_ack_count
```

The number of ACKs from a remote DataReader received by a local DataWriter.

9.40.2.18 received_ack_count_change

```
DDS_LongLong DDS_DataWriterProtocolStatus::received_ack_count_change
```

The change in **DDS_DataWriterProtocolStatus::received_ack_count** (p. 903) since the last time the status was read.

9.40.2.19 received_ack_bytes

DDS_LongLong DDS_DataWriterProtocolStatus::received_ack_bytes

The number of bytes of ACKs from a remote DataReader received by a local DataWriter.

9.40.2.20 received_ack_bytes_change

DDS_LongLong DDS_DataWriterProtocolStatus::received_ack_bytes_change

The change in **DDS_DataWriterProtocolStatus::received_ack_bytes** (p. 903) since the last time the status was read.

9.40.2.21 received_nack_count

DDS_LongLong DDS_DataWriterProtocolStatus::received_nack_count

The number of NACKs from a remote DataReader received by a local DataWriter.

9.40.2.22 received_nack_count_change

DDS_LongLong DDS_DataWriterProtocolStatus::received_nack_count_change

The change in **DDS_DataWriterProtocolStatus::received_nack_count** (p. 904) since the last time the status was read.

9.40.2.23 received_nack_bytes

DDS_LongLong DDS_DataWriterProtocolStatus::received_nack_bytes

The number of bytes of NACKs from a remote DataReader received by a local DataWriter.

9.40.2.24 received_nack_bytes_change

DDS_LongLong DDS_DataWriterProtocolStatus::received_nack_bytes_change

The change in **DDS_DataWriterProtocolStatus::received_nack_bytes** (p. 904) since the last time the status was read.

9.40.2.25 sent_gap_count

DDS_LongLong DDS_DataWriterProtocolStatus::sent_gap_count

The number of GAPs sent from local DataWriter to matching remote DataReaders.

9.40.2.26 sent_gap_count_change

DDS_LongLong DDS_DataWriterProtocolStatus::sent_gap_count_change

The change in **DDS_DataWriterProtocolStatus::sent_gap_count** (p. 904) since the last time the status was read.

9.40.2.27 sent_gap_bytes

DDS_LongLong DDS_DataWriterProtocolStatus::sent_gap_bytes

The number of bytes of GAPs sent from local DataWriter to matching remote DataReaders.

9.40.2.28 sent_gap_bytes_change

DDS_LongLong DDS_DataWriterProtocolStatus::sent_gap_bytes_change

The change in **DDS_DataWriterProtocolStatus::sent_gap_bytes** (p. 905) since the last time the status was read.

9.40.2.29 rejected_sample_count

DDS_LongLong DDS_DataWriterProtocolStatus::rejected_sample_count

[Not supported.]

9.40.2.30 rejected_sample_count_change

DDS_LongLong DDS_DataWriterProtocolStatus::rejected_sample_count_change

[Not supported.]

9.40.2.31 send_window_size

DDS_Long DDS_DataWriterProtocolStatus::send_window_size

Current maximum number of outstanding samples allowed in the DataWriter's queue.

Spans the range from **DDS_RtpsReliableWriterProtocol_t::min_send_window_size** (p. 1333) to **DDS_Rtps↩ ReliableWriterProtocol_t::max_send_window_size** (p. 1333).

9.40.2.32 first_available_sample_sequence_number

struct **DDS_SequenceNumber_t** DDS_DataWriterProtocolStatus::first_available_sample_sequence_number

The sequence number of the first available sample currently queued in the local DataWriter.

Applies only for local DataWriter status.

9.40.2.33 last_available_sample_sequence_number

struct **DDS_SequenceNumber_t** DDS_DataWriterProtocolStatus::last_available_sample_sequence_number

The sequence number of the last available sample currently queued in the local DataWriter.

Applies only for local DataWriter status.

9.40.2.34 first_unacknowledged_sample_sequence_number

struct **DDS_SequenceNumber_t** DDS_DataWriterProtocolStatus::first_unacknowledged_sample_sequence_↩ number

The sequence number of the first unacknowledged sample currently queued in the local DataWriter.

Applies only for local DataWriter status.

9.40.2.35 first_available_sample_virtual_sequence_number

struct **DDS_SequenceNumber_t** DDS_DataWriterProtocolStatus::first_available_sample_virtual_sequence↩ _number

The virtual sequence number of the first available sample currently queued in the local DataWriter.

Applies only for local DataWriter status.

9.40.2.36 last_available_sample_virtual_sequence_number

struct **DDS_SequenceNumber_t** DDS_DataWriterProtocolStatus::last_available_sample_virtual_sequence_number

The virtual sequence number of the last available sample currently queued in the local DataWriter.

Applies only for local DataWriter status.

9.40.2.37 first_unacknowledged_sample_virtual_sequence_number

struct **DDS_SequenceNumber_t** DDS_DataWriterProtocolStatus::first_unacknowledged_sample_virtual_sequence_number

The virtual sequence number of the first unacknowledged sample currently queued in the local DataWriter.

Applies only for local DataWriter status.

9.40.2.38 first_unacknowledged_sample_subscription_handle

DDS_InstanceHandle_t DDS_DataWriterProtocolStatus::first_unacknowledged_sample_subscription_handle

The handle of a remote DataReader that has not acknowledged the first unacknowledged sample of the local DataWriter.

Applies only for local DataWriter status.

9.40.2.39 first_unelapsed_keep_duration_sample_sequence_number

struct **DDS_SequenceNumber_t** DDS_DataWriterProtocolStatus::first_unelapsed_keep_duration_sample_sequence_number

The sequence number of the first sample whose keep duration has not yet elapsed.

Applicable only when **DDS_DataWriterProtocolQosPolicy::disable_positive_acks** (p. 894) is set.

Sequence number of the first sample kept in the DataWriter's queue whose keep_duration (applied when **DDS_DataWriterProtocolQosPolicy::disable_positive_acks** (p. 894) is set) has not yet elapsed.

Applies only for local DataWriter status.

9.40.2.40 pushed_fragment_count

DDS_LongLong DDS_DataWriterProtocolStatus::pushed_fragment_count

The number of DATA_FRAG messages that have been pushed by this DataWriter.

This statistic is incremented as each DATA_FRAG message is sent, not when the entire sample has been sent. Applicable only when data is fragmented.

9.40.2.41 pushed_fragment_bytes

DDS_LongLong DDS_DataWriterProtocolStatus::pushed_fragment_bytes

The number of bytes of DATA_FRAG messages that have been pushed by this DataWriter.

This statistic is incremented as each DATA_FRAG message is sent, not when the entire sample has been sent. Applicable only when data is fragmented.

9.40.2.42 pulled_fragment_count

DDS_LongLong DDS_DataWriterProtocolStatus::pulled_fragment_count

The number of DATA_FRAG messages that have been pulled from this DataWriter.

This statistic is incremented as each DATA_FRAG message is sent, not when the entire sample has been sent. Applicable only when data is fragmented.

9.40.2.43 pulled_fragment_bytes

DDS_LongLong DDS_DataWriterProtocolStatus::pulled_fragment_bytes

The number of bytes of DATA_FRAG messages that have been pulled from this DataWriter.

This statistic is incremented as each DATA_FRAG message is sent, not when the entire sample has been sent. Applicable only when data is fragmented.

9.40.2.44 received_nack_fragment_count

DDS_LongLong DDS_DataWriterProtocolStatus::received_nack_fragment_count

The number of NACK_FRAG messages that have been received by this DataWriter.

NACK FRAG RTPS messages are sent when large data is used in conjunction with reliable communication. They have the same properties as NACK messages, but instead of applying to samples, they apply to fragments. Applicable only when data is fragmented.

9.40.2.45 received_nack_fragment_bytes

DDS_LongLong DDS_DataWriterProtocolStatus::received_nack_fragment_bytes

The number of bytes of NACK_FRAG messages that have been received by this DataWriter.

NACK FRAG RTPS messages are sent when large data is used in conjunction with reliable communication. They have the same properties as NACK messages, but instead of applying to samples, they apply to fragments. Applicable only when data is fragmented.

9.41 DDS_DataWriterQos Struct Reference

QoS policies supported by a **DDSDataWriter** (p. 1589) entity.

Public Member Functions

- bool **operator==** (const **DDS_DataWriterQos** &r) const

  Compares two DataWriterQos objects for equality.
- bool **operator!=** (const **DDS_DataWriterQos** &r) const

  Compares two DataWriterQos objects for inequality.
- **DDS_ReturnCode_t print** () const

  *Prints this **DDS_DataWriterQos** (p. 909) to stdout.*
- **DDS_ReturnCode_t to_string** (char *string, **DDS_UnsignedLong** &string_size) const

  *Obtains a string representation of this **DDS_DataWriterQos** (p. 909).*
- **DDS_ReturnCode_t to_string** (char *string, **DDS_UnsignedLong** &string_size, const **DDS_DataWriterQos** &base) const

  *Obtains a string representation of this **DDS_DataWriterQos** (p. 909).*
- **DDS_ReturnCode_t to_string** (char *string, **DDS_UnsignedLong** &string_size, const **DDS_QosPrintFormat** &format) const

  *Obtains a string representation of this **DDS_DataWriterQos** (p. 909).*
- **DDS_ReturnCode_t to_string** (char *string, **DDS_UnsignedLong** &string_size, const **DDS_DataWriterQos** &base, const **DDS_QosPrintFormat** &format) const

  *Obtains a string representation of this **DDS_DataWriterQos** (p. 909).*
- **DDS_ReturnCode_t to_string** (char *string, **DDS_UnsignedLong** &string_size, const **DDS_QosPrintAll_t** &) const

  *Obtains a string representation of this **DDS_DataWriterQos** (p. 909).*
- **DDS_ReturnCode_t to_string** (char *string, **DDS_UnsignedLong** &string_size, const **DDS_QosPrintAll_t** &, const **DDS_QosPrintFormat** &format) const

  *Obtains a string representation of this **DDS_DataWriterQos** (p. 909).*

Public Attributes

- struct **DDS_DurabilityQosPolicy durability**

  *Durability policy, **DURABILITY** (p. 596).*
- struct **DDS_DurabilityServiceQosPolicy durability_service**

  *DurabilityService policy, **DURABILITY_SERVICE** (p. 600).*
- struct **DDS_DeadlineQosPolicy deadline**

  *<<**basic**>> (p. 201) Deadline policy, **DEADLINE** (p. 583).*
- struct **DDS_LatencyBudgetQosPolicy latency_budget**

  *Latency budget policy, **LATENCY_BUDGET** (p. 605).*
- struct **DDS_LivelinessQosPolicy liveliness**

  *Liveliness policy, **LIVELINESS** (p. 607).*
- struct **DDS_ReliabilityQosPolicy reliability**

  *Reliability policy, **RELIABILITY** (p. 631).*
- struct **DDS_DestinationOrderQosPolicy destination_order**

  *Destination order policy, **DESTINATION_ORDER** (p. 584).*

- struct **DDS_HistoryQosPolicy history**

  *<<**basic**>> (p. 201) History policy, **HISTORY** (p. 603).*
- struct **DDS_ResourceLimitsQosPolicy resource_limits**

  *Resource limits policy, **RESOURCE_LIMITS** (p. 635).*
- struct **DDS_TransportPriorityQosPolicy transport_priority**

  *Transport priority policy, **TRANSPORT_PRIORITY** (p. 647).*
- struct **DDS_LifespanQosPolicy lifespan**

  *<<**basic**>> (p. 201) Lifespan policy, **LIFESPAN** (p. 606).*
- struct **DDS_UserDataQosPolicy user_data**

  *User data policy, **USER_DATA** (p. 653).*
- struct **DDS_OwnershipQosPolicy ownership**

  *<<**basic**>> (p. 201) Ownership policy, **OWNERSHIP** (p. 612).*
- struct **DDS_OwnershipStrengthQosPolicy ownership_strength**

  *<<**basic**>> (p. 201) Ownership strength policy, **OWNERSHIP_STRENGTH** (p. 613).*
- struct **DDS_WriterDataLifecycleQosPolicy writer_data_lifecycle**

  *Writer data lifecycle policy, **WRITER_DATA_LIFECYCLE** (p. 654).*
- struct **DDS_DataRepresentationQosPolicy representation**

  *Data representation policy, **DATA_REPRESENTATION** (p. 567).*
- **DDS_DataTagQosPolicy data_tags**

  *DataTag policy, **DATA_TAG** (p. 574).*
- struct **DDS_DataWriterResourceLimitsQosPolicy writer_resource_limits**

  *<<**extension**>> (p. 432) Writer resource limits policy, **DATA_WRITER_RESOURCE_LIMITS** (p. 580).*
- struct **DDS_DataWriterProtocolQosPolicy protocol**

  *<<**extension**>> (p. 432) **DDSDataWriter** (p. 1589) protocol policy, **DATA_WRITER_PROTOCOL** (p. 579)*
- struct **DDS_TransportSelectionQosPolicy transport_selection**

  *<<**extension**>> (p. 432) Transport plugin selection policy, **TRANSPORT_SELECTION** (p. 648).*
- struct **DDS_TransportUnicastQosPolicy unicast**

  *<<**extension**>> (p. 432) Unicast transport policy, **TRANSPORT_UNICAST** (p. 649).*
- struct **DDS_PublishModeQosPolicy publish_mode**

  *<<**extension**>> (p. 432) Publish mode policy, **PUBLISH_MODE** (p. 627).*
- struct **DDS_PropertyQosPolicy property**

  *<<**basic**>> (p. 201) <<**extension**>> (p. 432) Property policy, **PROPERTY** (p. 617). See also `Property Reference Guide`.*
- struct **DDS_ServiceQosPolicy service**

  *<<**extension**>> (p. 432) Service policy, **SERVICE** (p. 636).*
- struct **DDS_BatchQosPolicy batch**

  *<<**extension**>> (p. 432) Batch policy, **BATCH** (p. 562).*
- struct **DDS_MultiChannelQosPolicy multi_channel**

  *<<**extension**>> (p. 432) Multi channel policy, **MULTICHANNEL** (p. 611).*
- struct **DDS_AvailabilityQosPolicy availability**

  *<<**extension**>> (p. 432) Availability policy, **AVAILABILITY** (p. 561).*
- struct **DDS_EntityNameQosPolicy publication_name**

  *<<**extension**>> (p. 432) EntityName policy, **ENTITY_NAME** (p. 601).*
- struct **DDS_TopicQueryDispatchQosPolicy topic_query_dispatch**

  *<<**extension**>> (p. 432) Topic Query dispatch policy, **TOPIC_QUERY_DISPATCH** (p. 640).*
- struct **DDS_DataWriterTransferModeQosPolicy transfer_mode**

  *<<**extension**>> (p. 432) TransferMode policy, **DATA_WRITER_TRANSFER_MODE** (p. 583).*
- struct **DDS_TypeSupportQosPolicy type_support**

  *<<**extension**>> (p. 432) Type support data, **TYPESUPPORT** (p. 651).*

9.41.1 Detailed Description

QoS policies supported by a **DDSDataWriter** (p. 1589) entity.

You must set certain members in a consistent manner:

- **DDS_DataWriterQos::history** (p. 913) .depth <= **DDS_DataWriterQos::resource_limits** (p. 913) .max_samples_per_instance
- **DDS_DataWriterQos::resource_limits** (p. 913) .max_samples_per_instance <= **DDS_DataWriterQos::resource_limits** (p. 913) .max_samples
- **DDS_DataWriterQos::resource_limits** (p. 913) .initial_samples <= **DDS_DataWriterQos::resource_limits** (p. 913) .max_samples
- **DDS_DataWriterQos::resource_limits** (p. 913) .initial_instances <= **DDS_DataWriterQos::resource_limits** (p. 913) .max_instances
- length of **DDS_DataWriterQos::user_data** (p. 914) .value <= **DDS_DomainParticipantQos::resource_limits** (p. 974) .writer_user_data_max_length

If any of the above are not true, **DDSDataWriter::set_qos** (p. 1604) and **DDSDataWriter::set_qos_with_profile** (p. 1605) and **DDSPublisher::set_default_datawriter_qos** (p. 1822) and **DDSPublisher::set_default_datawriter_qos_with_profile** (p. 1823) will fail with **DDS_RETCODE_INCONSISTENT_POLICY** (p. 533) and **DDSPublisher::create_datawriter** (p. 1826) and **DDSPublisher::create_datawriter_with_profile** (p. 1827) and will return NULL.

Entity:

DDSDataWriter (p. 1589)

See also

All QoS Policies (p. 549) allowed ranges within each Qos.

9.41.2 Member Function Documentation

9.41.2.1 operator==()

```
bool DDS_DataWriterQos::operator== (
            const  DDS_DataWriterQos & r ) const  [inline]
```

Compares two DataWriterQos objects for equality.

See also

DDS_DataWriterQos_equals (p. 117)

References **DDS_DataWriterQos_equals()**.

9.41.2.2 operator"!=()

```
bool DDS_DataWriterQos::operator!= (
            const  DDS_DataWriterQos & r ) const  [inline]
```

Compares two DataWriterQos objects for inequality.

See also

DDS_DataWriterQos_equals (p. 117)

References **DDS_DataWriterQos_equals()**.

9.41.3 Member Data Documentation

9.41.3.1 durability

```
struct  DDS_DurabilityQosPolicy DDS_DataWriterQos::durability
```

Durability policy, **DURABILITY** (p. 596).

9.41.3.2 durability_service

```
struct  DDS_DurabilityServiceQosPolicy DDS_DataWriterQos::durability_service
```

DurabilityService policy, **DURABILITY_SERVICE** (p. 600).

9.41.3.3 deadline

```
struct  DDS_DeadlineQosPolicy DDS_DataWriterQos::deadline
```

<<***basic***>> *(p. 201)* Deadline policy, **DEADLINE** (p. 583).

9.41.3.4 latency_budget

`struct` **DDS_LatencyBudgetQosPolicy** `DDS_DataWriterQos::latency_budget`

Latency budget policy, **LATENCY_BUDGET** (p. 605).

9.41.3.5 liveliness

`struct` **DDS_LivelinessQosPolicy** `DDS_DataWriterQos::liveliness`

Liveliness policy, **LIVELINESS** (p. 607).

9.41.3.6 reliability

`struct` **DDS_ReliabilityQosPolicy** `DDS_DataWriterQos::reliability`

Reliability policy, **RELIABILITY** (p. 631).

9.41.3.7 destination_order

`struct` **DDS_DestinationOrderQosPolicy** `DDS_DataWriterQos::destination_order`

Destination order policy, **DESTINATION_ORDER** (p. 584).

9.41.3.8 history

`struct` **DDS_HistoryQosPolicy** `DDS_DataWriterQos::history`

<<***basic***>> *(p. 201)* History policy, **HISTORY** (p. 603).

9.41.3.9 resource_limits

`struct` **DDS_ResourceLimitsQosPolicy** `DDS_DataWriterQos::resource_limits`

Resource limits policy, **RESOURCE_LIMITS** (p. 635).

9.41.3.10 transport_priority

`struct` **DDS_TransportPriorityQosPolicy** DDS_DataWriterQos::transport_priority

Transport priority policy, **TRANSPORT_PRIORITY** (p. 647).

9.41.3.11 lifespan

`struct` **DDS_LifespanQosPolicy** DDS_DataWriterQos::lifespan

<<***basic***>> *(p. 201)* Lifespan policy, **LIFESPAN** (p. 606).

9.41.3.12 user_data

`struct` **DDS_UserDataQosPolicy** DDS_DataWriterQos::user_data

User data policy, **USER_DATA** (p. 653).

9.41.3.13 ownership

`struct` **DDS_OwnershipQosPolicy** DDS_DataWriterQos::ownership

<<***basic***>> *(p. 201)* Ownership policy, **OWNERSHIP** (p. 612).

9.41.3.14 ownership_strength

`struct` **DDS_OwnershipStrengthQosPolicy** DDS_DataWriterQos::ownership_strength

<<***basic***>> *(p. 201)* Ownership strength policy, **OWNERSHIP_STRENGTH** (p. 613).

9.41.3.15 writer_data_lifecycle

`struct` **DDS_WriterDataLifecycleQosPolicy** DDS_DataWriterQos::writer_data_lifecycle

Writer data lifecycle policy, **WRITER_DATA_LIFECYCLE** (p. 654).

9.41.3.16 representation

`struct` **DDS_DataRepresentationQosPolicy** DDS_DataWriterQos::representation

Data representation policy, **DATA_REPRESENTATION** (p. 567).

9.41.3.17 data_tags

DDS_DataTagQosPolicy DDS_DataWriterQos::data_tags

DataTag policy, **DATA_TAG** (p. 574).

9.41.3.18 writer_resource_limits

`struct` **DDS_DataWriterResourceLimitsQosPolicy** DDS_DataWriterQos::writer_resource_limits

<<***extension***>> *(p. 432)* Writer resource limits policy, **DATA_WRITER_RESOURCE_LIMITS** (p. 580).

9.41.3.19 protocol

`struct` **DDS_DataWriterProtocolQosPolicy** DDS_DataWriterQos::protocol

<<***extension***>> *(p. 432)* **DDSDataWriter** (p. 1589) protocol policy, **DATA_WRITER_PROTOCOL** (p. 579)

9.41.3.20 transport_selection

`struct` **DDS_TransportSelectionQosPolicy** DDS_DataWriterQos::transport_selection

<<***extension***>> *(p. 432)* Transport plugin selection policy, **TRANSPORT_SELECTION** (p. 648).

Specifies the transports available for use by the **DDSDataWriter** (p. 1589).

9.41.3.21 unicast

`struct` **DDS_TransportUnicastQosPolicy** DDS_DataWriterQos::unicast

<<***extension***>> *(p. 432)* Unicast transport policy, **TRANSPORT_UNICAST** (p. 649).

Specifies the unicast transport interfaces and ports on which **messages** can be received.

The unicast interfaces are used to receive messages from **DDSDataReader** (p. 1556) entities in the domain.

9.41.3.22 publish_mode

`struct` **DDS_PublishModeQosPolicy** DDS_DataWriterQos::publish_mode

<<***extension***>> *(p. 432)* Publish mode policy, **PUBLISH_MODE** (p. 627).

Determines whether the **DDSDataWriter** (p. 1589) publishes data synchronously or asynchronously and how.

9.41.3.23 property

`struct` **DDS_PropertyQosPolicy** DDS_DataWriterQos::property

<<***basic***>> *(p. 201)* <<***extension***>> *(p. 432)* Property policy, **PROPERTY** (p. 617). See also `Property Reference Guide`.

9.41.3.24 service

`struct` **DDS_ServiceQosPolicy** DDS_DataWriterQos::service

<<***extension***>> *(p. 432)* Service policy, **SERVICE** (p. 636).

9.41.3.25 batch

`struct` **DDS_BatchQosPolicy** DDS_DataWriterQos::batch

<<***extension***>> *(p. 432)* Batch policy, **BATCH** (p. 562).

9.41.3.26 multi_channel

`struct` **DDS_MultiChannelQosPolicy** DDS_DataWriterQos::multi_channel

<<***extension***>> *(p. 432)* Multi channel policy, **MULTICHANNEL** (p. 611).

9.41.3.27 availability

`struct` **DDS_AvailabilityQosPolicy** DDS_DataWriterQos::availability

<<***extension***>> *(p. 432)* Availability policy, **AVAILABILITY** (p. 561).

9.41.3.28 publication_name

`struct` **`DDS_EntityNameQosPolicy`** `DDS_DataWriterQos::publication_name`

<<***extension***>> *(p. 432)* EntityName policy, **ENTITY_NAME** (p. 601).

9.41.3.29 topic_query_dispatch

`struct` **`DDS_TopicQueryDispatchQosPolicy`** `DDS_DataWriterQos::topic_query_dispatch`

<<***extension***>> *(p. 432)* Topic Query dispatch policy, **TOPIC_QUERY_DISPATCH** (p. 640).

9.41.3.30 transfer_mode

`struct` **`DDS_DataWriterTransferModeQosPolicy`** `DDS_DataWriterQos::transfer_mode`

<<***extension***>> *(p. 432)* TransferMode policy, **DATA_WRITER_TRANSFER_MODE** (p. 583).

9.41.3.31 type_support

`struct` **`DDS_TypeSupportQosPolicy`** `DDS_DataWriterQos::type_support`

<<***extension***>> *(p. 432)* Type support data, **TYPESUPPORT** (p. 651).

Optional value that is passed to a type plugin's on_endpoint_attached and serialization functions.

9.42 DDS_DataWriterResourceLimitsQosPolicy Struct Reference

<<***extension***>> *(p. 213)* Various settings that configure how a **DDSDataWriter** (p. 1589) allocates and uses physical memory for internal resources.

Public Attributes

- **DDS_Long initial_concurrent_blocking_threads**

  [DEPRECATED] *(p. **384**)* *<<**extension**>>* *(p. 213) The initial number of threads that are allowed to concurrently block on write call on the same **DDSDataWriter** (p. 1589).*
- **DDS_Long max_concurrent_blocking_threads**

  [DEPRECATED] *(p. **384**)* *<<**extension**>>* *(p. 213) The maximum number of threads that are allowed to concurrently block on write call on the same **DDSDataWriter** (p. 1589).*
- **DDS_Long max_remote_reader_filters**

  *<<**extension**>>* *(p. 213) The maximum number of remote DataReaders for which the **DDSDataWriter** (p. 1589) will perform content-based filtering.*
- **DDS_Long initial_batches**

  *<<**extension**>>* *(p. 213) Represents the initial number of batches a **DDSDataWriter** (p. 1589) will manage.*
- **DDS_Long max_batches**

  *<<**extension**>>* *(p. 213) Represents the maximum number of batches a **DDSDataWriter** (p. 1589) will manage.*
- **DDS_DataWriterResourceLimitsInstanceReplacementKind instance_replacement**

  *<<**extension**>>* *(p. 213) Sets the kinds of instances allowed to be replaced when instance resource limits are reached.*
- **DDS_Boolean replace_empty_instances**

  *<<**extension**>>* *(p. 213) Whether or not to replace empty instances during instance replacement*
- **DDS_Boolean autoregister_instances**

  [DEPRECATED] *(p. **384**)* *<<**extension**>>* *(p. 213) Whether or not to automatically register new instances*
- **DDS_Long initial_virtual_writers**

  *<<**extension**>>* *(p. 213) The initial number of virtual writers supported by a **DDSDataWriter** (p. 1589)*
- **DDS_Long max_virtual_writers**

  *<<**extension**>>* *(p. 213) The maximum number of virtual writers supported by a **DDSDataWriter** (p. 1589)*
- **DDS_Long max_remote_readers**

  *<<**extension**>>* *(p. 213) The maximum number of remote readers supported by a **DDSDataWriter** (p. 1589)*
- **DDS_Long max_app_ack_remote_readers**

  *<<**extension**>>* *(p. 213) The maximum number of application-level acknowledging remote readers supported by a **DDSDataWriter** (p. 1589)*
- **DDS_Long initial_active_topic_queries**

  *<<**extension**>>* *(p. 213) Represents the initial number of active topic queries a **DDSDataWriter** (p. 1589) will manage.*
- **DDS_Long max_active_topic_queries**

  *<<**extension**>>* *(p. 213) Represents the maximum number of active topic queries a **DDSDataWriter** (p. 1589) will manage.*
- struct **DDS_AllocationSettings_t writer_loaned_sample_allocation**

  *<<**extension**>>* *(p. 213) Represents the allocation settings of loaned samples managed by a **DDSDataWriter** (p. 1589).*
- **DDS_Boolean initialize_writer_loaned_sample**

  *<<**extension**>>* *(p. 213) Whether or not to initialize loaned samples returned by a **DDSDataWriter** (p. 1589)*
- **DDS_Long initial_remote_readers**

  *<<**extension**>>* *(p. 213) Controls the initial memory allocation for the matched remote readers of a **DDSDataWriter** (p. 1589).*

9.42.1 Detailed Description

<<***extension***>> *(p. 213)* Various settings that configure how a **DDSDataWriter** (p. 1589) allocates and uses physical memory for internal resources.

DataWriters must allocate internal structures to handle the simultaneously blocking of threads trying to call **FooDataWriter::write** (p. 1951) on the same **DDSDataWriter** (p. 1589), for the storage used to batch small samples, and for content-based filters specified by DataReaders.

Most of these internal structures start at an initial size and, by default, will be grown as needed by dynamically allocating additional memory. You may set fixed, maximum sizes for these internal structures if you want to bound the amount of memory that can be used by a **DDSDataWriter** (p. 1589). By setting the initial size to the maximum size, you will prevent RTI Connext from dynamically allocating any memory after the creation of the **DDSDataWriter** (p. 1589).

This QoS policy is an extension to the DDS standard.

Entity:

DDSDataWriter (p. 1589)

Properties:

RxO (p. 556) = N/A
Changeable (p. 556) = **NO** (p. 556)

9.42.2 Member Data Documentation

9.42.2.1 initial_concurrent_blocking_threads

```
DDS_Long DDS_DataWriterResourceLimitsQosPolicy::initial_concurrent_blocking_threads
```

[DEPRECATED] (p. 384) <<***extension***>> *(p. 213)* The initial number of threads that are allowed to concurrently block on write call on the same **DDSDataWriter** (p. 1589).

This value only applies if **DDS_HistoryQosPolicy** (p. 1162) has its kind set to **DDS_KEEP_ALL_HISTORY_QOS** (p. 604) and **DDS_ReliabilityQosPolicy::max_blocking_time** (p. 1298) is > 0.

[default] 1

[range] [1, 10000], <= max_concurrent_blocking_threads

Categories

Immutable (p. 333), **Middleware Memory Management** (p. 297), **Concurrency** (p. 258)

9.42.2.2 max_concurrent_blocking_threads

```
DDS_Long DDS_DataWriterResourceLimitsQosPolicy::max_concurrent_blocking_threads
```

[DEPRECATED] (p. **384)** <<***extension***>> *(*p. *213)* The maximum number of threads that are allowed to concurrently block on write call on the same **DDSDataWriter** (p. 1589).

This value only applies if **DDS_HistoryQosPolicy** (p. 1162) has its kind set to **DDS_KEEP_ALL_HISTORY_QOS** (p. 604) and **DDS_ReliabilityQosPolicy::max_blocking_time** (p. 1298) is > 0.

[default] DDS_LENGTH_UNLIMITED (p. 636)

[range] [1, 10000] or **DDS_LENGTH_UNLIMITED** (p. 636), >= initial_concurrent_blocking_threads

Categories

Immutable (p. 333), **Middleware Memory Management** (p. 297), **Concurrency** (p. 258)

9.42.2.3 max_remote_reader_filters

```
DDS_Long DDS_DataWriterResourceLimitsQosPolicy::max_remote_reader_filters
```

<<***extension***>> *(*p. *213)* The maximum number of remote DataReaders for which the **DDSDataWriter** (p. 1589) will perform content-based filtering.

[default] DDS_LENGTH_UNLIMITED (p. 636)

[range] [0, (2^31)-2] or **DDS_LENGTH_UNLIMITED** (p. 636).

0: The **DDSDataWriter** (p. 1589) will not perform filtering for any **DDSDataReader** (p. 1556).

1 to (2^31)-2: The DataWriter will filter for up to the specified number of DataReaders. In addition, the Datawriter will store the result of the filtering per sample per DataReader.

DDS_LENGTH_UNLIMITED (p. 636): The DataWriter will filter for up to (2^31)-2 DataReaders. However, in this case, the DataWriter will not store the filtering result per sample per DataReader. Thus, if a sample is resent (such as due to a loss of reliable communication), the sample will be filtered again.

Categories

Immutable (p. 333), **Middleware Memory Management** (p. 297), **Content Filtering** (p. 260)

9.42.2.4 initial_batches

```
DDS_Long DDS_DataWriterResourceLimitsQosPolicy::initial_batches
```

<<***extension***>> *(p. 213)* Represents the initial number of batches a **DDSDataWriter** (p. 1589) will manage.

[default] 8

[range] [1,100 million]

See also

DDS_BatchQosPolicy (p. 810)

Categories

Immutable (p. 333), **Sample Memory Management** (p. 321), **Middleware Memory Management** (p. 297), **Batching** (p. 257)

9.42.2.5 max_batches

```
DDS_Long DDS_DataWriterResourceLimitsQosPolicy::max_batches
```

<<***extension***>> *(p. 213)* Represents the maximum number of batches a **DDSDataWriter** (p. 1589) will manage.

[default] DDS_LENGTH_UNLIMITED (p. 636)

When batching is enabled, the maximum number of samples that a **DDSDataWriter** (p. 1589) can store is limited by this value and **DDS_ResourceLimitsQosPolicy::max_samples** (p. 1311).

[range] [1,100 million] or **DDS_LENGTH_UNLIMITED** (p. 636) >= **DDS_RtpsReliableWriterProtocol_t↩ ::heartbeats_per_max_samples** (p. 1327) if batching is enabled

See also

DDS_BatchQosPolicy (p. 810)

Categories

Immutable (p. 333), **Sample Memory Management** (p. 321), **Middleware Memory Management** (p. 297), **Batching** (p. 257)

9.42.2.6 instance_replacement

DDS_DataWriterResourceLimitsInstanceReplacementKind DDS_DataWriterResourceLimitsQosPolicy::instance↩
_replacement

<<***extension***>> *(p. 213)* Sets the kinds of instances allowed to be replaced when instance resource limits are reached.

When a **DDSDataWriter** (p. 1589)'s number of active instances is greater than **DDS_ResourceLimitsQosPolicy**↩**::max_instances** (p. 1311), it will try to make room by replacing an existing instance. This field specifies the kinds of instances allowed to be replaced.

If a replaceable instance is not available, either an out-of-resources exception will be returned, or the writer may block if the instance reclamation was done when writing.

[default] DDS_UNREGISTERED_INSTANCE_REPLACEMENT (p. 581)

See also

DDS_DataWriterResourceLimitsInstanceReplacementKind (p. 580)

Categories

Immutable (p. 333), **Middleware Memory Management** (p. 297), **Instance Lifecycle** (p. 286)

9.42.2.7 replace_empty_instances

DDS_Boolean DDS_DataWriterResourceLimitsQosPolicy::replace_empty_instances

<<***extension***>> *(p. 213)* Whether or not to replace empty instances during instance replacement

When a **DDSDataWriter** (p. 1589) has more active instances than allowed by **DDS_ResourceLimitsQosPolicy::max**↩**_instances** (p. 1311), it tries to make room by replacing an existing instance. This field configures whether empty instances (i.e. instances with no samples) may be replaced. If set **DDS_BOOLEAN_TRUE** (p. 513), then a **DDSData**↩**Writer** (p. 1589) will first try reclaiming empty instances, before trying to replace whatever is specified by **DDS_Data**↩**WriterResourceLimitsQosPolicy::instance_replacement** (p. 921).

[default] DDS_BOOLEAN_FALSE (p. 513)

See also

DDS_DataWriterResourceLimitsInstanceReplacementKind (p. 580)

Categories

Immutable (p. 333), **Middleware Memory Management** (p. 297), **Instance Lifecycle** (p. 286)

9.42.2.8 autoregister_instances

```
DDS_Boolean DDS_DataWriterResourceLimitsQosPolicy::autoregister_instances
```

[DEPRECATED] (p. **384)** <<***extension***>> *(*p. *213)* Whether or not to automatically register new instances

[default] DDS_BOOLEAN_FALSE (p. 513)

When set to true, it is possible to write with a non-NIL handle of an instance that is not registered: the write operation will succeed and the instance will be registered. Otherwise, that write operation would fail.

See also

FooDataWriter::write (p. 1951)

Categories

Immutable (p. 333), **Instance Lifecycle** (p. 286)

9.42.2.9 initial_virtual_writers

```
DDS_Long DDS_DataWriterResourceLimitsQosPolicy::initial_virtual_writers
```

<<***extension***>> *(*p. *213)* The initial number of virtual writers supported by a **DDSDataWriter** (p. 1589)

[default] 1

[range] [1, 1000000], or **DDS_LENGTH_UNLIMITED** (p. 636)

Categories

Immutable (p. 333), **Middleware Memory Management** (p. 297)

9.42.2.10 max_virtual_writers

```
DDS_Long DDS_DataWriterResourceLimitsQosPolicy::max_virtual_writers
```

<<***extension***>> *(*p. *213)* The maximum number of virtual writers supported by a **DDSDataWriter** (p. 1589)

Sets the maximum number of unique virtual writers supported by a **DDSDataWriter** (p. 1589), where virtual writers are added when samples are written with the virtual writer GUID.

This field is specially relevant in the configuration of Persistence Service DataWriters since these DataWriters will publish samples on behalf of multiple virtual writers.

[default] DDS_LENGTH_UNLIMITED (p. 636)

[range] [1, 1000000], or **DDS_LENGTH_UNLIMITED** (p. 636)

Categories

Immutable (p. 333), **Middleware Memory Management** (p. 297)

9.42.2.11 max_remote_readers

```
DDS_Long DDS_DataWriterResourceLimitsQosPolicy::max_remote_readers
```

<<***extension***>> *(p. 213)* The maximum number of remote readers supported by a **DDSDataWriter** (p. 1589)

[default] DDS_LENGTH_UNLIMITED (p. 636)

[range] [1, 1000000], or **DDS_LENGTH_UNLIMITED** (p. 636), >= initial_remote_readers

Categories

Immutable (p. 333), **Middleware Memory Management** (p. 297)

9.42.2.12 max_app_ack_remote_readers

```
DDS_Long DDS_DataWriterResourceLimitsQosPolicy::max_app_ack_remote_readers
```

<<***extension***>> *(p. 213)* The maximum number of application-level acknowledging remote readers supported by a **DDSDataWriter** (p. 1589)

[default] DDS_LENGTH_UNLIMITED (p. 636)

[range] [1, 1000000], or **DDS_LENGTH_UNLIMITED** (p. 636)

Categories

Immutable (p. 333), **Middleware Memory Management** (p. 297), **Application Acknowledgement** (p. 253), **Reliability** (p. 314)

9.42.2.13 initial_active_topic_queries

```
DDS_Long DDS_DataWriterResourceLimitsQosPolicy::initial_active_topic_queries
```

<<***extension***>> *(p. 213)* Represents the initial number of active topic queries a **DDSDataWriter** (p. 1589) will manage.

[default] 1

[range] [1, 1000000]

See also

DDS_TopicQueryDispatchQosPolicy (p. 1404)

Categories

Immutable (p. 333), **Middleware Memory Management** (p. 297), **Topic Queries** (p. 330)

9.42.2.14 max_active_topic_queries

DDS_Long DDS_DataWriterResourceLimitsQosPolicy::max_active_topic_queries

<<***extension***>> *(p. 213)* Represents the maximum number of active topic queries a **DDSDataWriter** (p. 1589) will manage.

When topic queries are enabled, the maximum number of topic queries that a **DDSDataWriter** (p. 1589) can publish data samples for at the same time is limited by this value.

When the DataWriter receives one topic query above this limit, it will wait to process it until it finishes publishing all the samples for at least one of the current topic queries.

[default] DDS_LENGTH_UNLIMITED (p. 636)

[range] [1, 1000000] or **DDS_LENGTH_UNLIMITED** (p. 636)

See also

DDS_TopicQueryDispatchQosPolicy (p. 1404)

Categories

Immutable (p. 333), **Middleware Memory Management** (p. 297), **Topic Queries** (p. 330)

9.42.2.15 writer_loaned_sample_allocation

struct **DDS_AllocationSettings_t** DDS_DataWriterResourceLimitsQosPolicy::writer_loaned_sample_↩allocation

<<***extension***>> *(p. 213)* Represents the allocation settings of loaned samples managed by a **DDSDataWriter** (p. 1589).

The number of samples loaned by a **DDSDataWriter** (p. 1589) via **FooDataWriter::get_loan** (p. 1963) is limited by the **DDS_AllocationSettings_t::max_count** (p. 797) of **DDS_DataWriterResourceLimitsQosPolicy::writer_loaned_↩sample_allocation** (p. 925). **FooDataWriter::get_loan** (p. 1963) returns NULL if and only if **DDS_Allocation↩Settings_t::max_count** (p. 797) samples have been loaned, and none of those samples has been written with **Foo↩DataWriter::write** (p. 1951) or discarded via **FooDataWriter::discard_loan** (p. 1964).

[default] initial_count = **DDS_AUTO_COUNT** (p. 595) (**DDS_ResourceLimitsQosPolicy::initial_samples** (p. 1312) + 1); max_count = **DDS_AUTO_COUNT** (p. 595) (**DDS_ResourceLimitsQosPolicy::max_samples** (p. 1311) + 1); incremental_count = **DDS_AUTO_COUNT** (p. 595) (0 if initial_count = max_count; initial_count otherwise);

[range] See allowed ranges in struct **DDS_AllocationSettings_t** (p. 796)

See also

FooDataWriter::get_loan (p. 1963)

FooDataWriter::discard_loan (p. 1964)

Categories

Immutable (p. 333), **Sample Memory Management** (p. 321), **Middleware Memory Management** (p. 297), **Sending Large Data** (p. 290)

9.42.2.16 initialize_writer_loaned_sample

```
DDS_Boolean DDS_DataWriterResourceLimitsQosPolicy::initialize_writer_loaned_sample
```

<<***extension***>> *(p. 213)* Whether or not to initialize loaned samples returned by a **DDSDataWriter** (p. 1589)

[default] DDS_BOOLEAN_FALSE (p. 513)

See also

FooDataWriter::get_loan (p. 1963)

Categories

Immutable (p. 333), **Sample Memory Management** (p. 321), **Sending Large Data** (p. 290)

9.42.2.17 initial_remote_readers

```
DDS_Long DDS_DataWriterResourceLimitsQosPolicy::initial_remote_readers
```

<<***extension***>> *(p. 213)* Controls the initial memory allocation for the matched remote readers of a **DDSDataWriter** (p. 1589).

The larger the value, the more memory is pre-allocated, the less memory is dynamically allocated.

[default] 2

[range] [0, 1000000], <= max_remote_readers

Categories

Immutable (p. 333), **Middleware Memory Management** (p. 297)

9.43 DDS_DataWriterShmemRefTransferModeSettings Struct Reference

Settings related to transferring data using shared memory references.

Public Attributes

- **DDS_Boolean enable_data_consistency_check**

  Controls if samples can be checked for consistency.

9.43.1 Detailed Description

Settings related to transferring data using shared memory references.

It is used to configure a **DDSDataWriter** (p. 1589) using **Zero Copy transfer over shared memory** (p. 73).

Properties:

RxO (p. 556) = N/A
Changeable (p. 556) = **NO** (p. 556)

QoS:

DDS_DataWriterTransferModeQosPolicy (p. 927)

9.43.2 Member Data Documentation

9.43.2.1 enable_data_consistency_check

```
DDS_Boolean DDS_DataWriterShmemRefTransferModeSettings::enable_data_consistency_check
```

Controls if samples can be checked for consistency.

When this setting is true, the **DDSDataWriter** (p. 1589) sends an incrementing sequence number as an inline QoS with every sample. This sequence number allows a **DDSDataReader** (p. 1556) to use the **FooDataReader::is_data_consistent** (p. 1943) API to detect if the **DDSDataWriter** (p. 1589) overwrote the sample before the **DDSDataReader** (p. 1556) could complete processing the sample.

[default] DDS_BOOLEAN_TRUE (p. 513)

9.44 DDS_DataWriterTransferModeQosPolicy Struct Reference

<<***extension***>> *(p. 213)* Qos related to transferring data

Public Attributes

- struct **DDS_DataWriterShmemRefTransferModeSettings** **shmem_ref_settings**
  <<***extension***>> *(p. 213) Settings related to transferring data using shared memory references.*

9.44.1 Detailed Description

<<***extension***>> *(p. 213)* Qos related to transferring data

It contains qualitative settings related to the actions a **DDSDataWriter** (p. 1589) performs while transferring its data.

Entity:

DDSDataWriter (p. 1589)

Properties:

RxO (p. 556) = N/A
Changeable (p. 556) = **NO** (p. 556)

9.44.2 Member Data Documentation

9.44.2.1 shmem_ref_settings

`struct` **`DDS_DataWriterShmemRefTransferModeSettings`** `DDS_DataWriterTransferModeQosPolicy::shmem_ref_settings`

<<***extension***>> *(p. 213)* Settings related to transferring data using shared memory references.

For details, refer to the **DDS_DataWriterShmemRefTransferModeSettings** (p. 926)

Categories

Immutable (p. 333), **Sending Large Data** (p. 290)

9.45 DDS_DeadlineQosPolicy Struct Reference

<<***spec***>> *(p. 209)* Expresses the maximum duration (deadline) within which an instance is expected to be updated.

Public Attributes

- struct **DDS_Duration_t period**

  <<***basic***>> *(p. 201)* <<***spec***>> *(p. 209) Duration of the deadline period.*

9.45.1 Detailed Description

<<***spec***>> *(p. 209)* Expresses the maximum duration (deadline) within which an instance is expected to be updated.

A **DDSDataReader** (p. 1556) expects a new sample updating the value of each instance at least once every `period`. That is, `period` specifies the maximum expected elapsed time between arriving data samples.

A **DDSDataWriter** (p. 1589) indicates that the application commits to write a new value (using the **DDSDataWriter** (p. 1589)) for each instance managed by the **DDSDataWriter** (p. 1589) at least once every `period`.

This QoS can be used during system integration to ensure that applications have been coded to meet design specifications.

It can also be used during runtime to detect when systems are performing outside of design specifications. Receiving applications can take appropriate actions to prevent total system failure when data is not received in time. For topics on which data is not expected to be periodic, `period` should be set to an infinite value.

Entity:

DDSTopic (p. 1885), **DDSDataReader** (p. 1556), **DDSDataWriter** (p. 1589)

Status:

DDS_OFFERED_DEADLINE_MISSED_STATUS (p. 539), **DDS_REQUESTED_DEADLINE_MISSED_STATUS** (p. 540), **DDS_OFFERED_INCOMPATIBLE_QOS_STATUS** (p. 540), **DDS_REQUESTED_INCOMPATIBLE_↩ QOS_STATUS** (p. 540)

Properties:

RxO (p. 556) = YES
Changeable (p. 556) = **YES** (p. 556)

9.45.2 Usage

This policy is useful for cases where a **DDSTopic** (p. 1885) is expected to have each instance updated periodically. On the publishing side this setting establishes a contract that the application must meet. On the subscribing side the setting establishes a minimum requirement for the remote publishers that are expected to supply the data values.

When RTI Connext 'matches' a **DDSDataWriter** (p. 1589) and a **DDSDataReader** (p. 1556) it checks whether the settings are compatible (i.e., *offered deadline* <= *requested deadline*); if they are not, the two entities are informed (via the **DDSListener** (p. 1794) or **DDSCondition** (p. 1544) mechanism) of the incompatibility of the QoS settings and communication will not occur.

Assuming that the reader and writer ends have compatible settings, the fulfilment of this contract is monitored by RTI Connext and the application is informed of any violations by means of the proper **DDSListener** (p. 1794) or **DDSCondition** (p. 1544).

9.45.3 Compatibility

The value offered is considered compatible with the value requested if and only if the inequality *offered period* <= *requested period* holds.

9.45.4 Consistency

The setting of the **DEADLINE** (p. 583) policy must be set consistently with that of the **TIME_BASED_FILTER** (p. 639).

For these two policies to be consistent the settings must be such that *deadline period* >= *minimum_separation*.

An attempt to set these policies in an inconsistent manner will result in **DDS_RETCODE_INCONSISTENT_POLICY** (p. 533) in **set_qos (abstract)** (p. 1732), or the **DDSEntity** (p. 1731) will not be created.

For a **DDSDataReader** (p. 1556), the **DEADLINE** (p. 583) policy and **DDS_TimeBasedFilterQosPolicy** (p. 1389) may interact such that even though the **DDSDataWriter** (p. 1589) is writing samples fast enough to fulfill its commitment to its own deadline, the **DDSDataReader** (p. 1556) may see violations of its deadline. This happens because RTI Connext will drop any samples received within the **DDS_TimeBasedFilterQosPolicy::minimum_separation** (p. 1391). To avoid triggering the **DDSDataReader** (p. 1556)'s deadline, even though the matched **DDSDataWriter** (p. 1589) is meeting its own deadline, set the two QoS parameters so that:

reader deadline >= *reader minimum_separation + writer deadline*

See **DDS_TimeBasedFilterQosPolicy** (p. 1389) for more information about the interactions between deadlines and time-based filters.

See also

DDS_TimeBasedFilterQosPolicy (p. 1389)

9.45.5 Member Data Documentation

9.45.5.1 period

`struct` **`DDS_Duration_t`** `DDS_DeadlineQosPolicy::period`

<<***basic***>> *(p. 201)* <<***spec***>> *(p. 209)* Duration of the deadline period.

[default] DDS_DURATION_INFINITE (p. 522)

[range] [1 nanosec, 1 year] or **DDS_DURATION_INFINITE** (p. 522), >= **DDS_TimeBasedFilterQosPolicy**↩ **::minimum_separation** (p. 1391)

Categories

Mutable (p. 369), **Discovery Metadata** (p. 377), **Matching** (p. 382), **Applied Per-Instance** (p. 375), **Data Availability** (p. 261)

9.46 DDS_DestinationOrderQosPolicy Struct Reference

<<***spec***>> *(p. 209)* Controls how the middleware will deal with data sent by multiple **DDSDataWriter** (p. 1589) entities for the same instance of data (i.e., same **DDSTopic** (p. 1885) and key).

Public Attributes

- **DDS_DestinationOrderQosPolicyKind kind**

  <<***basic***>> *(p. 201)* <<***spec***>> *(p. 209) Specifies the desired kind of destination order.*
- **DDS_DestinationOrderQosPolicyScopeKind scope**

  <<***spec***>> *(p. 209) Specifies the desired scope of the source destination order.*
- struct **DDS_Duration_t source_timestamp_tolerance**

  <<***extension***>> *(p. 213) Allowed tolerance between source timestamps of consecutive samples.*

9.46.1 Detailed Description

<<***spec***>> *(p. 209)* Controls how the middleware will deal with data sent by multiple **DDSDataWriter** (p. 1589) entities for the same instance of data (i.e., same **DDSTopic** (p. 1885) and key).

Entity:

DDSTopic (p. 1885), **DDSDataReader** (p. 1556), **DDSDataWriter** (p. 1589)

Status:

DDS_OFFERED_INCOMPATIBLE_QOS_STATUS (p. 540), **DDS_REQUESTED_INCOMPATIBLE_QOS_STATUS** (p. 540)

Properties:

RxO (p. 556) = YES
Changeable (p. 556) = **UNTIL ENABLE** (p. 556)

9.46.2 Usage

When multiple DataWriters send data for the same topic, the order in which data from different DataWriters are received by the applications of different DataReaders may be different. So different DataReaders may not receive the same "last" value when DataWriters stop sending data.

This QoS policy controls how each subscriber resolves the final value of a data instance that is written by multiple **DDSDataWriter** (p. 1589) entities (which may be associated with different **DDSPublisher** (p. 1818) entities) running on different nodes.

This QoS can be used to create systems that have the property of "eventual consistency." Thus intermediate states across multiple applications may be inconsistent, but when DataWriters stop sending changes to the same topic, all applications will end up having the same state.

This QoS policy can be set for both DataWriters and DataReaders.

For the DataReader:

The default setting, **DDS_BY_RECEPTION_TIMESTAMP_DESTINATIONORDER_QOS** (p. 585), indicates that (assuming the **OWNERSHIP_STRENGTH** (p. 613) policy allows it) the latest received value for the instance should be the one whose value is kept. That is, data will be delivered by a **DDSDataReader** (p. 1556) in the order in which it was *received* (which may lead to inconsistent final values).

For **DDS_BY_SOURCE_TIMESTAMP_DESTINATIONORDER_QOS** (p. 585), if the scope is set to **DDS_INSTANCE_SCOPE_DESTINATIONORDER_QOS** (p. 586) (default), within each instance, the sample's source timestamp shall be used to determine the most recent information. This is the only setting that, in the case of concurrent same-strength DataWriters updating the same instance, ensures that all DataReaders end up with the same final value for the instance. If a DataReader receives a sample for an instance with a source timestamp that is older than the last source timestamp received for the instance, the sample is dropped. The SAMPLE_REJECTED status or the SAMPLE_LOST status will not be updated.

If scope is set to **DDS_TOPIC_SCOPE_DESTINATIONORDER_QOS** (p. 586), the ordering is enforced per topic across all instances.

In addition, a DataReader will accept a sample only if the source timestamp is no farther in the future from the reception timestamp than source_timestamp_tolerance. Otherwise, the DDS sample is dropped. The SAMPLE_REJECTED status or the SAMPLE_LOST status will not be updated.

For the DataWriter:

For the default setting, **DDS_BY_RECEPTION_TIMESTAMP_DESTINATIONORDER_QOS** (p. 585), the DataWriter will not enforce source timestamp ordering when writing samples using the **FooDataWriter::write_w_params** (p. 1956) or **FooDataWriter::write_w_timestamp** (p. 1955) API. The source timestamp of a new sample can be older than the source timestamp of the previous samples.

When using **DDS_BY_SOURCE_TIMESTAMP_DESTINATIONORDER_QOS** (p. 585), If scope is set to **DDS_INSTANCE_SCOPE_DESTINATIONORDER_QOS** (p. 586) (default), when writing a sample, the sample's timestamp must not be older than the timestamp of the previously written DDS sample for the same instance. If, however, the timestamp is older than the timestamp of the previously written DDS sample—but the difference is less than the source_timestamp_tolerance—the DDS sample will use the previously written DDS sample's timestamp as its timestamp. Otherwise, if the difference is greater than the tolerance, the write will fail with retcode **DDS_RETCODE_BAD_PARAMETER** (p. 532).

If scope is set to **DDS_TOPIC_SCOPE_DESTINATIONORDER_QOS** (p. 586), a new sample timestamp must not be older than the timestamp of the previously written DDS sample, across all instances. (The ordering is enforced across all instances.)

9.46.3 Compatibility

The value offered is considered compatible with the value requested if and only if the inequality *offered kind >= requested kind* evaluates to 'TRUE'. For the purposes of this inequality, the values of **DDS_DestinationOrderQosPolicy::kind** (p. 933) are considered ordered such that **DDS_BY_RECEPTION_TIMESTAMP_DESTINATIONORDER_QOS** (p. 585) < **DDS_BY_SOURCE_TIMESTAMP_DESTINATIONORDER_QOS** (p. 585)

9.46.4 Member Data Documentation

9.46.4.1 kind

DDS_DestinationOrderQosPolicyKind DDS_DestinationOrderQosPolicy::kind

<<***basic***>> *(*p. *201)* <<***spec***>> *(*p. *209)* Specifies the desired kind of destination order.

[default] DDS_BY_RECEPTION_TIMESTAMP_DESTINATIONORDER_QOS (p. 585),

Categories

Immutable (p. 333), **Matching** (p. 382), **Discovery Metadata** (p. 377), **Applied Per-Instance** (p. 375), **Sample Ordering** (p. 324)

9.46.4.2 scope

DDS_DestinationOrderQosPolicyScopeKind DDS_DestinationOrderQosPolicy::scope

<<***spec***>> *(*p. *209)* Specifies the desired scope of the source destination order.

Indicates if tolerance check and the current sample's timestamp is computed based on instance or topic basis.
[default] DDS_INSTANCE_SCOPE_DESTINATIONORDER_QOS (p. 586)

Categories

Immutable (p. 333), **Applied Per-Instance** (p. 375), **Sample Ordering** (p. 324)

9.46.4.3 source_timestamp_tolerance

struct **DDS_Duration_t** DDS_DestinationOrderQosPolicy::source_timestamp_tolerance

<<***extension***>> *(*p. *213)* Allowed tolerance between source timestamps of consecutive samples.

When a **DDSDataWriter** (p. 1589) sets **DDS_DestinationOrderQosPolicyKind** (p. 585) to **DDS_BY_SOURCE_↩ TIMESTAMP_DESTINATIONORDER_QOS** (p. 585), when writing a sample, its timestamp must not be less than the timestamp of the previously written sample. However, if it is less than the timestamp of the previously written sample but the difference is less than this tolerance, the sample will use the previously written sample's timestamp as its timestamp. Otherwise, if the difference is greater than this tolerance, the write will fail.

When a **DDSDataReader** (p. 1556) sets **DDS_DestinationOrderQosPolicyKind** (p. 585) to **DDS_BY_SOURCE_↩ TIMESTAMP_DESTINATIONORDER_QOS** (p. 585), the **DDSDataReader** (p. 1556) will accept a sample only if the source timestamp is no farther in the future from the reception timestamp than this tolerance. Otherwise, the sample is dropped.

[default] 100 milliseconds for **DDSDataWriter** (p. 1589), 30 seconds for **DDSDataReader** (p. 1556)

Categories

Immutable (p. 333), **Applied Per-Instance** (p. 375), **Sample Ordering** (p. 324)

9.47 DDS_DiscoveryConfigQosPolicy Struct Reference

Settings for discovery configuration.

Public Attributes

- struct **DDS_Duration_t participant_liveliness_lease_duration**

  *<<**basic**>> (p. 201) <<**extension**>> (p. 213) The liveliness lease duration for the participant.*
- struct **DDS_Duration_t participant_liveliness_assert_period**

  *<<**basic**>> (p. 201) <<**extension**>> (p. 213) The period to assert liveliness for the participant.*
- struct **DDS_Duration_t participant_announcement_period**

  *<<**basic**>> (p. 201) <<**extension**>> (p. 213) The period at which a participant announces itself to potential peers when using the Simple Participant Discovery Protocol 2.0 (SPDP2).*
- **DDS_RemoteParticipantPurgeKind remote_participant_purge_kind**

  ***[DEPRECATED]** (p. **384**) <<**extension**>> (p. 213) The participant's behavior for maintaining knowledge of remote participants (and their contained entities) with which discovery communication has been lost.*
- struct **DDS_Duration_t max_liveliness_loss_detection_period**

  ***[DEPRECATED]** (p. **384**) <<**extension**>> (p. 213) The maximum amount of time between when a remote entity stops maintaining its liveliness and when the matched local entity realizes that fact.*
- **DDS_Long initial_participant_announcements**

  *<<**basic**>> (p. 201) <<**extension**>> (p. 213) The number of initial announcements sent when a participant is first enabled.*
- **DDS_Long new_remote_participant_announcements**

  *<<**basic**>> (p. 201) <<**extension**>> (p. 213) The number of participant announcements sent when a remote participant is newly discovered.*
- struct **DDS_Duration_t min_initial_participant_announcement_period**

  *<<**basic**>> (p. 201) <<**extension**>> (p. 213) The minimum period between initial announcements when a participant is first enabled or when a remote participant is newly discovered.*
- struct **DDS_Duration_t max_initial_participant_announcement_period**

  *<<**basic**>> (p. 201) <<**extension**>> (p. 213) The maximum period between initial announcements when a participant is first enabled or when a remote participant is newly discovered.*
- struct **DDS_BuiltinTopicReaderResourceLimits_t participant_reader_resource_limits**

  *<<**extension**>> (p. 213) Resource limits.*
- struct **DDS_RtpsReliableReaderProtocol_t publication_reader**

  *<<**extension**>> (p. 213) RTPS protocol-related configuration settings for the RTPS reliable reader associated with a built-in publication reader.*
- struct **DDS_BuiltinTopicReaderResourceLimits_t publication_reader_resource_limits**

  *<<**extension**>> (p. 213) Resource limits.*
- struct **DDS_RtpsReliableReaderProtocol_t subscription_reader**

  *<<**extension**>> (p. 213) RTPS protocol-related configuration settings for the RTPS reliable reader associated with a built-in subscription reader.*
- struct **DDS_BuiltinTopicReaderResourceLimits_t subscription_reader_resource_limits**

  *<<**extension**>> (p. 213) Resource limits.*
- struct **DDS_RtpsReliableWriterProtocol_t publication_writer**

  *<<**extension**>> (p. 213) RTPS protocol-related configuration settings for the RTPS reliable writer associated with a built-in publication writer.*
- struct **DDS_WriterDataLifecycleQosPolicy publication_writer_data_lifecycle**

  *<<**extension**>> (p. 213) Writer data lifecycle settings for a built-in publication writer.*

- struct **DDS_RtpsReliableWriterProtocol_t subscription_writer**

  *<<**extension**>> (p. 213) RTPS protocol-related configuration settings for the RTPS reliable writer associated with a built-in subscription writer.*
- struct **DDS_WriterDataLifecycleQosPolicy subscription_writer_data_lifecycle**

  *<<**extension**>> (p. 213) Writer data lifecycle settings for a built-in subscription writer.*
- **DDS_DiscoveryConfigBuiltinPluginKindMask builtin_discovery_plugins**

  *<<**extension**>> (p. 213) Mask of built-in discovery plugin kinds.*
- **DDS_DiscoveryConfigBuiltinChannelKindMask enabled_builtin_channels**

  *<<**extension**>> (p. 213) The mask specifying which built-in channels should be enabled.*
- **DDS_ReliabilityQosPolicyKind participant_message_reader_reliability_kind**

  ***[DEPRECATED]** (p. **384**) <<**extension**>> (p. 213) Reliability policy for a built-in participant message reader.*
- struct **DDS_RtpsReliableReaderProtocol_t participant_message_reader**

  *<<**extension**>> (p. 213) RTPS reliable reader protocol-related configuration settings for a built-in participant message reader. This parameter only has effect if **DDS_DiscoveryConfigQosPolicy::participant_message_reader_reliability_kind** (p. 946) is set to **DDS_RELIABLE_RELIABILITY_QOS** (p. 633).*
- struct **DDS_RtpsReliableWriterProtocol_t participant_message_writer**

  *<<**extension**>> (p. 213) RTPS protocol-related configuration settings for the RTPS reliable writer associated with the built-in participant message writer. This parameter only has effect if the matching participant message reader is configured with **DDS_RELIABLE_RELIABILITY_QOS** (p. 633) **DDS_ReliabilityQosPolicyKind** (p. 632).*
- struct **DDS_PublishModeQosPolicy publication_writer_publish_mode**

  *<<**extension**>> (p. 213) Publish mode policy for the built-in publication writer.*
- struct **DDS_PublishModeQosPolicy subscription_writer_publish_mode**

  *<<**extension**>> (p. 213) Publish mode policy for the built-in subscription writer.*
- struct **DDS_AsynchronousPublisherQosPolicy asynchronous_publisher**

  *<<**extension**>> (p. 213) Asynchronous publishing settings for the discovery **DDSPublisher** (p. 1818) and all entities that are created by it.*
- struct **DDS_Duration_t default_domain_announcement_period**

  *<<**extension**>> (p. 213) The period to announce a participant to the default domain 0.*
- **DDS_Boolean ignore_default_domain_announcements**

  ***[DEPRECATED]** (p. **384**) <<**extension**>> (p. 213) Used to ignore the announcements received by a participant on the default domain 0 corresponding to participants running on domains IDs other than 0.*
- struct **DDS_RtpsReliableWriterProtocol_t service_request_writer**

  *<<**extension**>> (p. 213) RTPS protocol-related configuration settings for the RTPS reliable writer associated with the built-in **DDS_ServiceRequest** (p. 1359) writer.*
- struct **DDS_WriterDataLifecycleQosPolicy service_request_writer_data_lifecycle**

  *<<**extension**>> (p. 213) Writer data lifecycle settings for a built-in **DDS_ServiceRequest** (p. 1359) writer.*
- struct **DDS_PublishModeQosPolicy service_request_writer_publish_mode**

  *<<**extension**>> (p. 213) Publish mode policy for the built-in service request writer.*
- struct **DDS_RtpsReliableReaderProtocol_t service_request_reader**

  *<<**extension**>> (p. 213) RTPS reliable reader protocol-related configuration settings for a built-in **DDS_ServiceRequest** (p. 1359) reader.*
- struct **DDS_Duration_t locator_reachability_assert_period**

  *<<**extension**>> (p. 213) Period at which this DomainParticipant will assert the locators discovered from other DomainParticipants.*
- struct **DDS_Duration_t locator_reachability_lease_duration**

  *<<**extension**>> (p. 213) The time period after which other DomainParticipants can consider one of their locators as "unreachable" if they do not receive a REACHABILITY PING from this DomainParticipant.*
- struct **DDS_Duration_t locator_reachability_change_detection_period**

  *<<**extension**>> (p. 213) Period at which this DomainParticipant will check if its locators are reachable from other DomainParticipants.*

- struct **DDS_RtpsReliableWriterProtocol_t secure_volatile_writer**

  *<<**extension**>> (p. 213) RTPS protocol-related configuration settings for the RTPS reliable writer associated with the built-in secure volatile writer.*
- struct **DDS_PublishModeQosPolicy secure_volatile_writer_publish_mode**

  *<<**extension**>> (p. 213) Publish mode policy for the built-in secure volatile writer.*
- struct **DDS_RtpsReliableReaderProtocol_t secure_volatile_reader**

  *<<**extension**>> (p. 213) RTPS reliable reader protocol-related configuration settings for the built-in secure volatile reader.*
- **DDS_Long endpoint_type_object_lb_serialization_threshold**

  *<<**extension**>> (p. 213) Controls when/if the TypeObject v1 is compressed when propagated in Simple Endpoint Discovery.*
- struct **DDS_Duration_t dns_tracker_polling_period**

  *<<**extension**>> (p. 213) Duration that specifies the period used by the DNS tracker to poll the DNS service and check for changes in the hostnames.*
- struct **DDS_PublishModeQosPolicy participant_configuration_writer_publish_mode**

  *<<**extension**>> (p. 213) Publish mode policy for the built-in participant configuration writer.*
- struct **DDS_RtpsReliableWriterProtocol_t participant_configuration_writer**

  *<<**extension**>> (p. 213) RTPS protocol-related configuration settings for the RTPS reliable writer associated with a built-in participant configuration writer.*
- struct **DDS_WriterDataLifecycleQosPolicy participant_configuration_writer_data_lifecycle**

  *<<**extension**>> (p. 213) Writer data lifecycle settings for a built-in participant configuration writer.*
- struct **DDS_RtpsReliableReaderProtocol_t participant_configuration_reader**

  *<<**extension**>> (p. 213) RTPS protocol-related configuration settings for the RTPS reliable reader associated with a built-in participant configuration reader.*
- struct **DDS_BuiltinTopicReaderResourceLimits_t participant_configuration_reader_resource_limits**

  *<<**extension**>> (p. 213) Resource limits for the built-in topic participant configuration reader.*
- DDS_String **request_types_filter**

  *<<**extension**>> (p. 213) Topic name pattern for requesting all types via Type Lookup Service.*

9.47.1 Detailed Description

Settings for discovery configuration.

*<<**extension**>> (p. 213)* This QoS policy controls the amount of delay in discovering entities in the system and the amount of discovery traffic in the network.

The amount of network traffic required by the discovery process can vary widely, based on how your application has chosen to configure the middleware's network addressing (e.g., unicast vs. multicast, multicast TTL, etc.), the size of the system, whether all applications are started at the same time or whether start times are staggered, and other factors. Your application can use this policy to make tradeoffs between discovery completion time and network bandwidth utilization. In addition, you can introduce random back-off periods into the discovery process to decrease the probability of network contention when many applications start simultaneously.

Entity:

DDSDomainParticipant (p. 1619)

Properties:

RxO (p. 556) = N/A
Changeable (p. 556) = **NO** (p. 556)

9.47.2 Member Data Documentation

9.47.2.1 participant_liveliness_lease_duration

`struct` **`DDS_Duration_t`** `DDS_DiscoveryConfigQosPolicy::participant_liveliness_lease_duration`

<<***basic***>> *(p. 201)* <<***extension***>> *(p. 213)* The liveliness lease duration for the participant.

This is the same as the expiration time of the DomainParticipant as defined in the RTPS protocol.

If the participant has not refreshed its own liveliness to other participants at least once within this period, it may be considered as stale by other participants in the network.

Should be strictly greater than **DDS_DiscoveryConfigQosPolicy::participant_liveliness_assert_period** (p. 937).

[default] 100 seconds

[range] [1 nanosec,1 year], > participant_liveliness_assert_period

Categories

Immutable (p. 333), **Discovery** (p. 271), **Liveliness** (p. 292), **Discovery Metadata** (p. 377)

9.47.2.2 participant_liveliness_assert_period

`struct` **`DDS_Duration_t`** `DDS_DiscoveryConfigQosPolicy::participant_liveliness_assert_period`

<<***basic***>> *(p. 201)* <<***extension***>> *(p. 213)* The period to assert liveliness for the participant.

The period at which the participant will refresh its liveliness to all the peers.

Should be strictly less than **DDS_DiscoveryConfigQosPolicy::participant_liveliness_lease_duration** (p. 937).

[default] 30 seconds

[range] [1 nanosec,1 year], < participant_liveliness_lease_duration

Categories

Immutable (p. 333), **Discovery** (p. 271), **Liveliness** (p. 292)

9.47.2.3 participant_announcement_period

`struct` **`DDS_Duration_t`** `DDS_DiscoveryConfigQosPolicy::participant_announcement_period`

*<<**basic**>> (p. 201) <<**extension**>> (p. 213)* The period at which a participant announces itself to potential peers when using the Simple Participant Discovery Protocol 2.0 (SPDP2).

The **DDS_DiscoveryQosPolicy::initial_peers** (p. 962) list **DDSDomainParticipant::add_peer** (p. 1678) API are used to configure a set of potential peers that a DomainParticipant may discover. The **DDS_DiscoveryConfigQosPolicy::participant_announcement_period** (p. 937) configures how frequently a DomainParticipant will announce itself to the subset of the configured potential peers that it has not matched with yet. Once a DomainParticipant matches with a DomainParticipant at one of configured potential peer locators, it will no longer announce itself to that locator at this period unless liveliness is lost.

This QoS policy is only supported when using the Simple Participant Discovery Protocol 2.0 (SPDP2). Setting this value when using the Simple Participant Discovery Protocol (SPDP) or other participant discovery protocols is not supported and will result in an error.

[default] DDS_DURATION_AUTO (p. 523) (Takes the value of **DDS_DiscoveryConfigQosPolicy::participant_liveliness_assert_period** (p. 937))

[range] [1 nanosec,1 year]

Categories

Immutable (p. 333), **Discovery** (p. 271)

9.47.2.4 remote_participant_purge_kind

`DDS_RemoteParticipantPurgeKind` `DDS_DiscoveryConfigQosPolicy::remote_participant_purge_kind`

[DEPRECATED] (p. 384) *<<**extension**>> (p. 213)* The participant's behavior for maintaining knowledge of remote participants (and their contained entities) with which discovery communication has been lost.

Most users will not need to change this value from its default, **DDS_LIVELINESS_BASED_REMOTE_PARTICIPANT_PURGE** (p. 593). However, **DDS_NO_REMOTE_PARTICIPANT_PURGE** (p. 594) may be a good choice if the following conditions apply:

1. Discovery communication with a remote participant may be lost while data communication remains intact. Such will not typically be the case if discovery takes place over the Simple Discovery Protocol, but may be the case if the RTI Enterprise Discovery Service is used.
2. Extensive and prolonged lack of discovery communication between participants is not expected to be common, either because participant loss itself is expected to be rare, or because participants may be lost sporadically but will typically return again.
3. Maintaining inter-participant liveliness is problematic, perhaps because a participant has no writers with the appropriate **DDS_LivelinessQosPolicyKind** (p. 607).

[default] DDS_LIVELINESS_BASED_REMOTE_PARTICIPANT_PURGE (p. 593)

Categories

Immutable (p. 333), **Discovery** (p. 271), **Liveliness** (p. 292)

9.47.2.5 max_liveliness_loss_detection_period

```
struct DDS_Duration_t DDS_DiscoveryConfigQosPolicy::max_liveliness_loss_detection_period
```

[DEPRECATED] (p. **384)** <<***extension***>> *(*p. *213)* The maximum amount of time between when a remote entity stops maintaining its liveliness and when the matched local entity realizes that fact.

Notification of the loss of liveliness of a remote entity may come more quickly than this duration, depending on the liveliness contract between the local and remote entities and the capabilities of the discovery mechanism in use. For example, a **DDSDataReader** (p. 1556) will learn of the loss of liveliness of a matched **DDSDataWriter** (p. 1589) within the reader's offered liveliness lease duration.

Shortening this duration will increase the responsiveness of entities to communication failures. However, it will also increase the CPU usage of the application, as the liveliness of remote entities will be examined more frequently.

[default] 60 seconds

[range] [1 nanosec, 1 year]

Categories

Immutable (p. 333), **Discovery** (p. 271), **Liveliness** (p. 292)

9.47.2.6 initial_participant_announcements

```
DDS_Long DDS_DiscoveryConfigQosPolicy::initial_participant_announcements
```

<<***basic***>> *(*p. *201)* <<***extension***>> *(*p. *213)* The number of initial announcements sent when a participant is first enabled.

[default] 5

[range] [1,1 million]

Categories

Immutable (p. 333), **Discovery** (p. 271)

9.47.2.7 new_remote_participant_announcements

`DDS_Long DDS_DiscoveryConfigQosPolicy::new_remote_participant_announcements`

*<<**basic**>> (p. 201) <<**extension**>> (p. 213)* The number of participant announcements sent when a remote participant is newly discovered.

These announcements are only sent to the newly discovered remote participant, they are not also broadcast to the initial_peers list.

[default] 2

[range] [0,1 million]

Categories

Immutable (p. 333), **Discovery** (p. 271)

9.47.2.8 min_initial_participant_announcement_period

`struct DDS_Duration_t DDS_DiscoveryConfigQosPolicy::min_initial_participant_announcement_period`

*<<**basic**>> (p. 201) <<**extension**>> (p. 213)* The minimum period between initial announcements when a participant is first enabled or when a remote participant is newly discovered.

A random delay between this and **DDS_DiscoveryConfigQosPolicy::max_initial_participant_announcement_period** (p. 940) is introduced in between initial announcements when a new remote participant is discovered.

The setting of **DDS_DiscoveryConfigQosPolicy::min_initial_participant_announcement_period** (p. 940) must be consistent with **DDS_DiscoveryConfigQosPolicy::max_initial_participant_announcement_period** (p. 940). For these two values to be consistent, they must verify that:

DDS_DiscoveryConfigQosPolicy::min_initial_participant_announcement_period (p. 940) <= **DDS_DiscoveryConfigQosPolicy::max_initial_participant_announcement_period** (p. 940).

[default] 10 milliseconds

[range] [1 nanosec,1 year]

Categories

Immutable (p. 333), **Discovery** (p. 271)

9.47.2.9 max_initial_participant_announcement_period

`struct` **`DDS_Duration_t`** `DDS_DiscoveryConfigQosPolicy::max_initial_participant_announcement_period`

*<<**basic**>> (p. 201) <<**extension**>> (p. 213)* The maximum period between initial announcements when a participant is first enabled or when a remote participant is newly discovered.

A random delay between **DDS_DiscoveryConfigQosPolicy::min_initial_participant_announcement_period** (p. 940) and this is introduced in between initial announcements when a new remote participant is discovered.

The setting of **DDS_DiscoveryConfigQosPolicy::max_initial_participant_announcement_period** (p. 940) must be consistent with **DDS_DiscoveryConfigQosPolicy::min_initial_participant_announcement_period** (p. 940). For these two values to be consistent, they must verify that:

DDS_DiscoveryConfigQosPolicy::min_initial_participant_announcement_period (p. 940) <= **DDS_Discovery↩ ConfigQosPolicy::max_initial_participant_announcement_period** (p. 940).

[default] 1 second

[range] [1 nanosec,1 year]

Categories

Immutable (p. 333), **Discovery** (p. 271)

9.47.2.10 participant_reader_resource_limits

`struct` **`DDS_BuiltinTopicReaderResourceLimits_t`** `DDS_DiscoveryConfigQosPolicy::participant_reader_↩ resource_limits`

*<<**extension**>> (p. 213)* Resource limits.

Resource limit of the built-in topic participant reader. For details, see **DDS_BuiltinTopicReaderResourceLimits_t** (p. 815).

Categories

Immutable (p. 333), **Discovery** (p. 271), **Sample Memory Management** (p. 321), **Middleware Memory Management** (p. 297)

9.47.2.11 publication_reader

`struct` **`DDS_RtpsReliableReaderProtocol_t`** `DDS_DiscoveryConfigQosPolicy::publication_reader`

<<***extension***>> *(p. 213)* RTPS protocol-related configuration settings for the RTPS reliable reader associated with a built-in publication reader.

For details, refer to the **DDS_RtpsReliableReaderProtocol_t** (p. 1314)

[default]
min_heartbeat_response_delay 0.0 seconds;
max_heartbeat_response_delay 0.0 seconds;
heartbeat_suppression_duration 0.0625 seconds;
nack_period 5.0 seconds;
receive_window_size 256;
round_trip_time 0.0 seconds;
app_ack_period 5.0 seconds;
samples_per_app_ack 1;

Categories

Immutable (p. 333), **Discovery** (p. 271), **Reliability** (p. 314)

9.47.2.12 publication_reader_resource_limits

`struct` **`DDS_BuiltinTopicReaderResourceLimits_t`** `DDS_DiscoveryConfigQosPolicy::publication_reader_↩ resource_limits`

<<***extension***>> *(p. 213)* Resource limits.

Resource limit of the built-in topic publication reader. For details, see **DDS_BuiltinTopicReaderResourceLimits_t** (p. 815).

Categories

Immutable (p. 333), **Discovery** (p. 271), **Sample Memory Management** (p. 321), **Middleware Memory Management** (p. 297)

9.47.2.13 subscription_reader

```
struct DDS_RtpsReliableReaderProtocol_t DDS_DiscoveryConfigQosPolicy::subscription_reader
```

<<***extension***>> *(p. 213)* RTPS protocol-related configuration settings for the RTPS reliable reader associated with a built-in subscription reader.

For details, refer to the **DDS_RtpsReliableReaderProtocol_t** (p. 1314)

[default]
min_heartbeat_response_delay 0.0 seconds;
max_heartbeat_response_delay 0.0 seconds;
heartbeat_suppression_duration 0.0625 seconds;
nack_period 5.0 seconds;
receive_window_size 256;
round_trip_time 0.0 seconds;
app_ack_period 5.0 seconds;
samples_per_app_ack 1;

Categories

Immutable (p. 333), **Discovery** (p. 271), **Reliability** (p. 314)

9.47.2.14 subscription_reader_resource_limits

```
struct DDS_BuiltinTopicReaderResourceLimits_t DDS_DiscoveryConfigQosPolicy::subscription_reader↩
_resource_limits
```

<<***extension***>> *(p. 213)* Resource limits.

Resource limit of the built-in topic subscription reader. For details, see **DDS_BuiltinTopicReaderResourceLimits_t** (p. 815).

Categories

Immutable (p. 333), **Discovery** (p. 271), **Sample Memory Management** (p. 321), **Middleware Memory Management** (p. 297)

9.47.2.15 publication_writer

`struct` **`DDS_RtpsReliableWriterProtocol_t`** `DDS_DiscoveryConfigQosPolicy::publication_writer`

<<***extension***>> *(p. 213)* RTPS protocol-related configuration settings for the RTPS reliable writer associated with a built-in publication writer.

For details, refer to the **DDS_RtpsReliableWriterProtocol_t** (p. 1320)

[default]
low_watermark 0;
high_watermark 1;
heartbeat_period 3.0 seconds;
fast_heartbeat_period 3.0 seconds;
late_joiner_heartbeat_period 3.0 seconds;
virtual_heartbeat_period **DDS_DURATION_INFINITE** (p. 522);
samples_per_virtual_heartbeat **DDS_LENGTH_UNLIMITED** (p. 636);
max_heartbeat_retries 10;
inactivate_nonprogressing_readers **DDS_BOOLEAN_FALSE** (p. 513);
heartbeats_per_max_samples 8;
min_nack_response_delay 0.0 seconds;
max_nack_response_delay 0.0 seconds;
nack_suppression_duration 0.0 seconds;
max_bytes_per_nack_response 131072 bytes;
disable_positive_acks_min_sample_keep_duration 1.0 milliseconds;
disable_positive_acks_max_sample_keep_duration 1.0 seconds;
disable_positive_acks_enable_adaptive_sample_keep_duration **DDS_BOOLEAN_TRUE** (p. 513);
disable_positive_acks_decrease_sample_keep_duration_factor 95;
disable_positive_acks_increase_sample_keep_duration_factor 150;
min_send_window_size **DDS_LENGTH_UNLIMITED** (p. 636);
max_send_window_size **DDS_LENGTH_UNLIMITED** (p. 636);
send_window_update_period 3s;
send_window_increase_factor 105;
send_window_decrease_factor 50;
enable_multicast_periodic_heartbeat **DDS_BOOLEAN_FALSE** (p. 513);
multicast_resend_threshold 2 readers;
disable_repair_piggyback_heartbeat **DDS_BOOLEAN_FALSE** (p. 513);

Categories

Immutable (p. 333), **Discovery** (p. 271), **Reliability** (p. 314)

9.47.2.16 publication_writer_data_lifecycle

`struct` **`DDS_WriterDataLifecycleQosPolicy`** `DDS_DiscoveryConfigQosPolicy::publication_writer_data_↩ lifecycle`

<<***extension***>> *(p. 213)* Writer data lifecycle settings for a built-in publication writer.

For details, refer to the **DDS_WriterDataLifecycleQosPolicy** (p. 1524). **DDS_WriterDataLifecycleQosPolicy↩ ::autodispose_unregistered_instances** (p. 1525) will always be forced to **DDS_BOOLEAN_TRUE** (p. 513).

Categories

Immutable (p. 333), **Discovery** (p. 271), **Instance Lifecycle** (p. 286)

9.47.2.17 subscription_writer

`struct` **`DDS_RtpsReliableWriterProtocol_t`** `DDS_DiscoveryConfigQosPolicy::subscription_writer`

<<***extension***>> *(p. 213)* RTPS protocol-related configuration settings for the RTPS reliable writer associated with a built-in subscription writer.

For details, refer to the **DDS_RtpsReliableWriterProtocol_t** (p. 1320)

[default]
low_watermark 0;
high_watermark 1;
heartbeat_period 3.0 seconds;
fast_heartbeat_period 3.0 seconds;
late_joiner_heartbeat_period 3.0 seconds;
virtual_heartbeat_period **DDS_DURATION_INFINITE** (p. 522);
samples_per_virtual_heartbeat **DDS_LENGTH_UNLIMITED** (p. 636);
max_heartbeat_retries 10;
inactivate_nonprogressing_readers **DDS_BOOLEAN_FALSE** (p. 513);
heartbeats_per_max_samples 8;
min_nack_response_delay 0.0 seconds;
max_nack_response_delay 0.0 seconds;
nack_suppression_duration 0.0 seconds;
max_bytes_per_nack_response 131072 bytes;
disable_positive_acks_min_sample_keep_duration 1.0 milliseconds;
disable_positive_acks_max_sample_keep_duration 1.0 seconds;
disable_positive_acks_enable_adaptive_sample_keep_duration **DDS_BOOLEAN_TRUE** (p. 513);
disable_positive_acks_decrease_sample_keep_duration_factor 95;
disable_positive_acks_increase_sample_keep_duration_factor 150;
min_send_window_size **DDS_LENGTH_UNLIMITED** (p. 636);
max_send_window_size **DDS_LENGTH_UNLIMITED** (p. 636);
send_window_update_period 3s;
send_window_increase_factor 105;
send_window_decrease_factor 50;
enable_multicast_periodic_heartbeat **DDS_BOOLEAN_FALSE** (p. 513);
multicast_resend_threshold 2 readers;
disable_repair_piggyback_heartbeat **DDS_BOOLEAN_FALSE** (p. 513);

Categories

Immutable (p. 333), **Discovery** (p. 271), **Reliability** (p. 314)

9.47.2.18 subscription_writer_data_lifecycle

`struct` **DDS_WriterDataLifecycleQosPolicy** `DDS_DiscoveryConfigQosPolicy::subscription_writer_data_↩ lifecycle`

<<***extension***>> *(p. 213)* Writer data lifecycle settings for a built-in subscription writer.

For details, refer to the **DDS_WriterDataLifecycleQosPolicy** (p. 1524). **DDS_WriterDataLifecycleQosPolicy↩ ::autodispose_unregistered_instances** (p. 1525) will always be forced to **DDS_BOOLEAN_TRUE** (p. 513).

Categories

Immutable (p. 333), **Discovery** (p. 271), **Instance Lifecycle** (p. 286)

9.47.2.19 builtin_discovery_plugins

DDS_DiscoveryConfigBuiltinPluginKindMask `DDS_DiscoveryConfigQosPolicy::builtin_discovery_plugins`

<<***extension***>> *(p. 213)* Mask of built-in discovery plugin kinds.

There are several built-in discovery plugins. This mask enables the different plugins. Any plugin not enabled will not be created.

[default] DDS_DISCOVERYCONFIG_BUILTIN_SDP (p. 591)

See also

DDS_DiscoveryConfigBuiltinPluginKind (p. 591)

Categories

Immutable (p. 333), **Discovery** (p. 271)

9.47.2.20 enabled_builtin_channels

DDS_DiscoveryConfigBuiltinChannelKindMask `DDS_DiscoveryConfigQosPolicy::enabled_builtin_channels`

<<***extension***>> *(p. 213)* The mask specifying which built-in channels should be enabled.

While there are a number of built-in channels that are used by Connext DDS, the only built-in channels which can currently be enabled or disabled are the Service Request Channel and the Type Lookup Service Channel.

If **DDS_DomainParticipantResourceLimitsQosPolicy::type_object_max_serialized_length** (p. 994) is set to 0 the middleware will automatically disable the **DDS_DISCOVERYCONFIG_TYPE_LOOKUP_SERVICE_CHANNEL** (p. 592) built-in channel.

[default] DDS_DISCOVERYCONFIG_SERVICE_REQUEST_CHANNEL (p. 592) | **DDS_DISCOVERYCONFIG_↩ TYPE_LOOKUP_SERVICE_CHANNEL** (p. 592)

Categories

Immutable (p. 333)

9.47.2.21 participant_message_reader_reliability_kind

DDS_ReliabilityQosPolicyKind DDS_DiscoveryConfigQosPolicy::participant_message_reader_reliability_kind

[DEPRECATED] (p. 384) <<***extension***>> *(p. 213)* Reliability policy for a built-in participant message reader.

For details, refer to the **DDS_ReliabilityQosPolicyKind** (p. 632).

[default] DDS_BEST_EFFORT_RELIABILITY_QOS (p. 633)

Categories

Immutable (p. 333), **Discovery** (p. 271), **Reliability** (p. 314), **Liveliness** (p. 292)

9.47.2.22 participant_message_reader

struct **DDS_RtpsReliableReaderProtocol_t** DDS_DiscoveryConfigQosPolicy::participant_message_reader

<<***extension***>> *(p. 213)* RTPS reliable reader protocol-related configuration settings for a built-in participant message reader. This parameter only has effect if **DDS_DiscoveryConfigQosPolicy::participant_message_reader_reliability_kind** (p. 946) is set to **DDS_RELIABLE_RELIABILITY_QOS** (p. 633).

For details, refer to the **DDS_RtpsReliableReaderProtocol_t** (p. 1314)

[default]
min_heartbeat_response_delay 0.0 seconds;
max_heartbeat_response_delay 0.0 seconds;
heartbeat_suppression_duration 0.0625 seconds;
nack_period 5.0 seconds;
receive_window_size 256;
round_trip_time 0.0 seconds;
app_ack_period 5.0 seconds;
samples_per_app_ack 1;

Categories

Immutable (p. 333), **Discovery** (p. 271), **Reliability** (p. 314), **Liveliness** (p. 292)

9.47.2.23 participant_message_writer

```
struct DDS_RtpsReliableWriterProtocol_t DDS_DiscoveryConfigQosPolicy::participant_message_writer
```

<<***extension***>> *(p. 213)* RTPS protocol-related configuration settings for the RTPS reliable writer associated with the built-in participant message writer. This parameter only has effect if the matching participant message reader is configured with **DDS_RELIABLE_RELIABILITY_QOS** (p. 633) **DDS_ReliabilityQosPolicyKind** (p. 632).

For details, refer to the **DDS_RtpsReliableWriterProtocol_t** (p. 1320)

[default]
low_watermark 0;
high_watermark 1;
heartbeat_period 1.0 seconds;
fast_heartbeat_period 1.0 seconds;
late_joiner_heartbeat_period 1.0 seconds;
virtual_heartbeat_period **DDS_DURATION_INFINITE** (p. 522);
samples_per_virtual_heartbeat **DDS_LENGTH_UNLIMITED** (p. 636);
max_heartbeat_retries 10;
inactivate_nonprogressing_readers **DDS_BOOLEAN_FALSE** (p. 513);
heartbeats_per_max_samples 1;
min_nack_response_delay 0.0 seconds;
max_nack_response_delay 0.0 seconds;
nack_suppression_duration 0.0 seconds;
max_bytes_per_nack_response 9216 bytes;
disable_positive_acks_min_sample_keep_duration 1.0 milliseconds;
disable_positive_acks_max_sample_keep_duration 1.0 seconds;
disable_positive_acks_enable_adaptive_sample_keep_duration **DDS_BOOLEAN_TRUE** (p. 513);
disable_positive_acks_decrease_sample_keep_duration_factor 95;
disable_positive_acks_increase_sample_keep_duration_factor 150;
min_send_window_size **DDS_LENGTH_UNLIMITED** (p. 636);
max_send_window_size **DDS_LENGTH_UNLIMITED** (p. 636);
send_window_update_period 1s;
send_window_increase_factor 105;
send_window_decrease_factor 50;
enable_multicast_periodic_heartbeat **DDS_BOOLEAN_FALSE** (p. 513);
multicast_resend_threshold 2 readers;
disable_repair_piggyback_heartbeat **DDS_BOOLEAN_FALSE** (p. 513);

Categories

Immutable (p. 333), **Discovery** (p. 271), **Reliability** (p. 314), **Liveliness** (p. 292)

9.47.2.24 publication_writer_publish_mode

```
struct DDS_PublishModeQosPolicy DDS_DiscoveryConfigQosPolicy::publication_writer_publish_mode
```

<<***extension***>> *(p. 213)* Publish mode policy for the built-in publication writer.

Determines whether the Discovery built-in publication **DDSDataWriter** (p. 1589) publishes data synchronously or asynchronously and how.

Categories

Immutable (p. 333), **Discovery** (p. 271), **Flow Control** (p. 282)

9.47.2.25 subscription_writer_publish_mode

`struct` **`DDS_PublishModeQosPolicy`** `DDS_DiscoveryConfigQosPolicy::subscription_writer_publish_mode`

<<***extension***>> *(p. 213)* Publish mode policy for the built-in subscription writer.

Determines whether the Discovery built-in subscription **DDSDataWriter** (p. 1589) publishes data synchronously or asynchronously and how.

Categories

Immutable (p. 333), **Discovery** (p. 271), **Flow Control** (p. 282)

9.47.2.26 asynchronous_publisher

`struct` **`DDS_AsynchronousPublisherQosPolicy`** `DDS_DiscoveryConfigQosPolicy::asynchronous_publisher`

<<***extension***>> *(p. 213)* Asynchronous publishing settings for the discovery **DDSPublisher** (p. 1818) and all entities that are created by it.

Categories

Immutable (p. 333), **Discovery** (p. 271), **Flow Control** (p. 282)

9.47.2.27 default_domain_announcement_period

`struct DDS_Duration_t DDS_DiscoveryConfigQosPolicy::default_domain_announcement_period`

<<***extension***>> *(p. 213)* The period to announce a participant to the default domain 0.

The period at which a participant will announce itself to the default domain 0 using the default UDPv4 multicast group address for discovery traffic on that domain.

For domain 0, the default discovery multicast address is 239.255.0.1:7400.

To disable announcement to the default domain, set this period to **DDS_DURATION_INFINITE** (p. 522).

When this period is set to a value other than **DDS_DURATION_INFINITE** (p. 522) and **DDS_DiscoveryConfigQos↩ Policy::ignore_default_domain_announcements** (p. 950) is set to **DDS_BOOLEAN_FALSE** (p. 513), you can get information about participants running in different domains by creating a participant in domain 0 and implementing the on_data_available callback in the **DDS_ParticipantBuiltinTopicData** (p. 1229) built-in DataReader's listener.

You can learn the domain ID associated with a participant by looking at the field **DDS_ParticipantBuiltinTopicData↩ ::domain_id** (p. 1232).

[default] 30 seconds

[range] [1 nanosec,1 year] or **DDS_DURATION_INFINITE** (p. 522)

See also

DDS_ParticipantBuiltinTopicData::domain_id (p. 1232)

DDS_DiscoveryConfigQosPolicy::ignore_default_domain_announcements (p. 950)

Categories

Immutable (p. 333), **Discovery** (p. 271), **Debuggability** (p. 266)

9.47.2.28 ignore_default_domain_announcements

`DDS_Boolean DDS_DiscoveryConfigQosPolicy::ignore_default_domain_announcements`

[DEPRECATED] (p. 384) <<***extension***>> *(p. 213)* Used to ignore the announcements received by a participant on the default domain 0 corresponding to participants running on domains IDs other than 0.

This setting only applies to participants running on the default domain 0 and using the default port mapping.

When this setting is set to **DDS_BOOLEAN_TRUE** (p. 513), a participant running on the default domain 0 will ignore announcements from participants running on different domain IDs.

When this setting is set to **DDS_BOOLEAN_FALSE** (p. 513), a participant running on the default domain 0 will provide announcements from participants running on different domain IDs to the application via the **DDS_ParticipantBuiltin↩ TopicData** (p. 1229) built-in DataReader.

[default] DDS_BOOLEAN_TRUE (p. 513)

See also

DDS_ParticipantBuiltinTopicData::domain_id (p. 1232)

DDS_DiscoveryConfigQosPolicy::default_domain_announcement_period (p. 949)

Categories

Immutable (p. 333), **Discovery** (p. 271), **Debuggability** (p. 266)

9.47.2.29 service_request_writer

`struct` **`DDS_RtpsReliableWriterProtocol_t`** `DDS_DiscoveryConfigQosPolicy::service_request_writer`

<<***extension***>> *(p. 213)* RTPS protocol-related configuration settings for the RTPS reliable writer associated with the built-in **DDS_ServiceRequest** (p. 1359) writer.

For details, refer to the **DDS_RtpsReliableWriterProtocol_t** (p. 1320)

[default]
low_watermark 0;
high_watermark 1;
heartbeat_period 3.0 seconds;
fast_heartbeat_period 3.0 seconds;
late_joiner_heartbeat_period 3.0 seconds;
virtual_heartbeat_period **DDS_DURATION_INFINITE** (p. 522);
samples_per_virtual_heartbeat **DDS_LENGTH_UNLIMITED** (p. 636);
max_heartbeat_retries 10;
inactivate_nonprogressing_readers **DDS_BOOLEAN_FALSE** (p. 513);
heartbeats_per_max_samples 8;
min_nack_response_delay 0.0 seconds;
max_nack_response_delay 0.0 seconds;
nack_suppression_duration 0.0 seconds;
max_bytes_per_nack_response 131072 bytes;
disable_positive_acks_min_sample_keep_duration 1.0 milliseconds;
disable_positive_acks_max_sample_keep_duration 1.0 seconds;
disable_positive_acks_enable_adaptive_sample_keep_duration **DDS_BOOLEAN_TRUE** (p. 513);
disable_positive_acks_decrease_sample_keep_duration_factor 95;
disable_positive_acks_increase_sample_keep_duration_factor 150;
min_send_window_size **DDS_LENGTH_UNLIMITED** (p. 636);
max_send_window_size **DDS_LENGTH_UNLIMITED** (p. 636);
send_window_update_period 3s;
send_window_increase_factor 105;
send_window_decrease_factor 50;
enable_multicast_periodic_heartbeat **DDS_BOOLEAN_FALSE** (p. 513);
multicast_resend_threshold 2 readers;
disable_repair_piggyback_heartbeat **DDS_BOOLEAN_FALSE** (p. 513);

Categories

Immutable (p. 333), **Discovery** (p. 271), **Reliability** (p. 314)

9.47.2.30 service_request_writer_data_lifecycle

`struct` **`DDS_WriterDataLifecycleQosPolicy`** `DDS_DiscoveryConfigQosPolicy::service_request_writer_↩ data_lifecycle`

<<***extension***>> *(p. 213)* Writer data lifecycle settings for a built-in **DDS_ServiceRequest** (p. 1359) writer.

For details, refer to the **DDS_WriterDataLifecycleQosPolicy** (p. 1524).

Categories

Immutable (p. 333), **Discovery** (p. 271), **Instance Lifecycle** (p. 286)

9.47.2.31 service_request_writer_publish_mode

`struct` **`DDS_PublishModeQosPolicy`** `DDS_DiscoveryConfigQosPolicy::service_request_writer_publish_mode`

<<***extension***>> *(p. 213)* Publish mode policy for the built-in service request writer.

Determines whether the Discovery built-in service request **DDSDataWriter** (p. 1589) publishes data synchronously or asynchronously and how.

Categories

Immutable (p. 333), **Discovery** (p. 271), **Flow Control** (p. 282)

9.47.2.32 service_request_reader

`struct` **`DDS_RtpsReliableReaderProtocol_t`** `DDS_DiscoveryConfigQosPolicy::service_request_reader`

<<***extension***>> *(p. 213)* RTPS reliable reader protocol-related configuration settings for a built-in **DDS_Service↩ Request** (p. 1359) reader.

For details, refer to the **DDS_RtpsReliableReaderProtocol_t** (p. 1314)

[default]
min_heartbeat_response_delay 0.0 seconds;
max_heartbeat_response_delay 0.0 seconds;
heartbeat_suppression_duration 0.0625 seconds;
nack_period 5.0 seconds;
receive_window_size 256;
round_trip_time 0.0 seconds;
app_ack_period 5.0 seconds;
samples_per_app_ack 1;

Categories

Immutable (p. 333), **Discovery** (p. 271), **Reliability** (p. 314)

9.47.2.33 locator_reachability_assert_period

```
struct DDS_Duration_t DDS_DiscoveryConfigQosPolicy::locator_reachability_assert_period
```

<<***extension***>> *(p. 213)* Period at which this DomainParticipant will assert the locators discovered from other DomainParticipants.

This setting configures the period at which this **DDSDomainParticipant** (p. 1619) will ping all the locators that it has discovered from other DomainParticipants. This period should be strictly less than **DDS_DiscoveryConfigQosPolicy::locator_reachability_lease_duration** (p. 953).

If **DDS_DiscoveryConfigQosPolicy::locator_reachability_lease_duration** (p. 953) is **DDS_DURATION_INFINITE** (p. 522) this parameter is ignored. The DomainParticipant will not assert remote locators.

[default] 20 seconds

[range] [1 nanosec,1 year]

See also

DDS_DiscoveryConfigQosPolicy::locator_reachability_lease_duration (p. 953)

Categories

Immutable (p. 333), **Transports** (p. 327)

9.47.2.34 locator_reachability_lease_duration

```
struct DDS_Duration_t DDS_DiscoveryConfigQosPolicy::locator_reachability_lease_duration
```

<<***extension***>> *(p. 213)* The time period after which other DomainParticipants can consider one of their locators as "unreachable" if they do not receive a REACHABILITY PING from this DomainParticipant.

For the purpose of this explanation, we will use 'local' to refer to the DomainParticipant in which we configure locator_reachability_lease_duration and 'remote' to refer to the other DomainParticipants communicating with the local DomainParticipant.

This setting configures a timeout announced to the remote DomainParticipants. This timeout is used by the remote DomainParticipants as the maximum period by which a remote locator must be asserted by the local DomainParticipant (through a REACHABILITY PING message) before considering this locator as "unreachable" from the local DomainParticipant.

When a remote DomainParticipant detects that one of its locators is not reachable from the local DomainParticipant, it will notify the local DomainParticipant of this event. From that moment on, and until notified otherwise, the local DomainParticipant will not send RTPS messages to remote DomainParticipants using this locator.

If this value is set to **DDS_DURATION_INFINITE** (p. 522), the local DomainParticipant will send RTPS messages to a remote DomainParticipant on the locators announced by the remote DomainParticipant, regardless of whether or not the remote DomainParticipant can be reached using these locators.

[default] DDS_DURATION_INFINITE (p. 522)

[range] [1 nanosec,1 year] or **DDS_DURATION_INFINITE** (p. 522)

9.47.2.35 locator_reachability_change_detection_period

```
struct DDS_Duration_t DDS_DiscoveryConfigQosPolicy::locator_reachability_change_detection_period
```

<<***extension***>> *(p. 213)* Period at which this DomainParticipant will check if its locators are reachable from other DomainParticipants.

This setting determines the maximum period at which this DomainParticipant will check to see if its locators are reachable from other DomainParticipants according to the other DomainParticipants' **DDS_DiscoveryConfigQosPolicy::locator_reachability_lease_duration** (p. 953) value.

If **DDS_DiscoveryConfigQosPolicy::locator_reachability_lease_duration** (p. 953) is **DDS_DURATION_INFINITE** (p. 522) this parameter is ignored. The DomainParticipant will not schedule an event to see if its locators are reachable from other DomainParticipants.

[default] 60 seconds

[range] [1 nanosec,1 year]

See also

DDS_DiscoveryConfigQosPolicy::locator_reachability_lease_duration (p. 953)

Categories

Immutable (p. 333), **Transports** (p. 327)

9.47.2.36 secure_volatile_writer

```
struct DDS_RtpsReliableWriterProtocol_t DDS_DiscoveryConfigQosPolicy::secure_volatile_writer
```

<<***extension***>> *(p. 213)* RTPS protocol-related configuration settings for the RTPS reliable writer associated with the built-in secure volatile writer.

For details, refer to the **DDS_RtpsReliableWriterProtocol_t** (p. 1320)

[default]
low_watermark 0;
high_watermark 1;
heartbeat_period 200.0 millisecond;
fast_heartbeat_period 200.0 milliseconds;
late_joiner_heartbeat_period 200.0 millisecond;
virtual_heartbeat_period **DDS_DURATION_INFINITE** (p. 522);
samples_per_virtual_heartbeat **DDS_LENGTH_UNLIMITED** (p. 636);
max_heartbeat_retries 1000;
inactivate_nonprogressing_readers **DDS_BOOLEAN_FALSE** (p. 513);
heartbeats_per_max_samples 100000000;
min_nack_response_delay 0.0 seconds;

max_nack_response_delay 0.0 seconds;
nack_suppression_duration 0.0 seconds;
max_bytes_per_nack_response 9216 bytes;
disable_positive_acks_min_sample_keep_duration 1.0 millisecond;
disable_positive_acks_max_sample_keep_duration 1.0 second;
disable_positive_acks_enable_adaptive_sample_keep_duration **DDS_BOOLEAN_TRUE** (p. 513);
disable_positive_acks_decrease_sample_keep_duration_factor 95;
disable_positive_acks_increase_sample_keep_duration_factor 150;
min_send_window_size **DDS_LENGTH_UNLIMITED** (p. 636);
max_send_window_size **DDS_LENGTH_UNLIMITED** (p. 636);
send_window_update_period 1.0 second;
send_window_increase_factor 105;
send_window_decrease_factor 50;
enable_multicast_periodic_heartbeat **DDS_BOOLEAN_FALSE** (p. 513);
multicast_resend_threshold 2 readers;
disable_repair_piggyback_heartbeat **DDS_BOOLEAN_FALSE** (p. 513);

Categories

Immutable (p. 333), **Discovery** (p. 271), **Reliability** (p. 314), **Security** (p. 325)

9.47.2.37 secure_volatile_writer_publish_mode

`struct` **`DDS_PublishModeQosPolicy`** `DDS_DiscoveryConfigQosPolicy::secure_volatile_writer_publish_mode`

<<***extension***>> *(p. 213)* Publish mode policy for the built-in secure volatile writer.

Determines whether the built-in secure volatile **DDSDataWriter** (p. 1589) publishes data synchronously or asynchronously and how.

Categories

Immutable (p. 333), **Discovery** (p. 271), **Flow Control** (p. 282), **Security** (p. 325)

9.47.2.38 secure_volatile_reader

`struct` **`DDS_RtpsReliableReaderProtocol_t`** `DDS_DiscoveryConfigQosPolicy::secure_volatile_reader`

<<***extension***>> *(p. 213)* RTPS reliable reader protocol-related configuration settings for the built-in secure volatile reader.

For details, refer to the **DDS_RtpsReliableReaderProtocol_t** (p. 1314)

[default]
min_heartbeat_response_delay 0.0 seconds;
max_heartbeat_response_delay 0.0 seconds;
heartbeat_suppression_duration 0.0625 seconds;
nack_period 5.0 seconds;
receive_window_size 256;
round_trip_time 0.0 seconds;
app_ack_period 5.0 seconds;
samples_per_app_ack 1;

Categories

Immutable (p. 333), **Discovery** (p. 271), **Reliability** (p. 314), **Security** (p. 325)

9.47.2.39 endpoint_type_object_lb_serialization_threshold

```
DDS_Long DDS_DiscoveryConfigQosPolicy::endpoint_type_object_lb_serialization_threshold
```

<<***extension***>> *(p. 213)* Controls when/if the TypeObject v1 is compressed when propagated in Simple Endpoint Discovery.

Minimum size (in bytes) of the serialized TypeObject v1 that will trigger the serialization of a compressed TypeObject v1 (known as TypeObjectLb) instead of the regular TypeObject v1.

For example, setting this property to 1000 will trigger the serialization of the TypeObjectLb for TypeObject v1s whose serialized size is greater than 1000 Bytes.

The sentinel value -1 disables TypeObject v1 compression.

If **DDS_DISCOVERYCONFIG_TYPE_LOOKUP_SERVICE_CHANNEL** (p. 592) is also enabled, this threshold does not affect compression of TypeObject v2s in TypeLookup Service messages.

[default] 0. The default value 0 enables TypeObject v1 compression by always sending TypeObjectLb.

[range] [-1, 2147483647]

Categories

Immutable (p. 333), **Discovery** (p. 271), **Sending Large Data** (p. 290), **Type System** (p. 331)

9.47.2.40 dns_tracker_polling_period

```
struct DDS_Duration_t DDS_DiscoveryConfigQosPolicy::dns_tracker_polling_period
```

<<***extension***>> *(p. 213)* Duration that specifies the period used by the DNS tracker to poll the DNS service and check for changes in the hostnames.

RTI Connext allows the use of hostnames instead of IP addresses when configuring initial peers for specific transports (e.g.: UDPv4 and UDPv6). The DNS tracker keeps the IP addresses of these hostnames updated. The DNS tracker builds a list of hostnames from the initial peers of a DomainParticipant, queries the DNS for those hostnames, and updates the resolved IP addresses when the IP addresses change. The frequency of these queries is defined by the DNS tracker polling period.

RTI Connext keeps information regarding the hostnames of peers if they are part of the **DDS_DiscoveryQos↩ Policy::initial_peers** (p. 962). The information regarding peers added through the **DDSDomainParticipant::add_peer** (p. 1678) operation is kept only if the DNS tracker has been enabled before adding a peer.

The DNS tracker is useful in situations where the IP address of an as-yet undiscovered peer may change. For example, if the **DDS_DiscoveryQosPolicy::initial_peers** (p. 962) of a participant is set to "hostname1" and the IP address associated with "hostname1" changes after the participant is created, the participant has no way of knowing about that change unless it queries the DNS again.

When two participants have already discovered each other, any IP address change is communicated directly between the participants and there is no need for the DNS tracker. In other words, they do not wait for the DNS tracker polling period to update the IP address–the change is communicated immediately between the participants so that communication can continue uninterrupted.

The value should be (but is not required to be) set to a value that is less than the **DDS_DiscoveryConfigQosPolicy↩ ::participant_liveliness_assert_period** (p. 937) (if using SPDP) or **DDS_DiscoveryConfigQosPolicy::participant↩ _liveliness_assert_period** (p. 937) (is using SPDP2) so that multiple participant announcements are not sent to an outdated IP address.

When the period is set to **DDS_DURATION_INFINITE** (p. 522), the tracker is disabled.

[default] 20 seconds

[range] [1 second,1 year], **DDS_DURATION_INFINITE** (p. 522)

Categories

Immutable (p. 333), **Discovery** (p. 271), **Transports** (p. 327)

9.47.2.41 participant_configuration_writer_publish_mode

`struct` **`DDS_PublishModeQosPolicy`** `DDS_DiscoveryConfigQosPolicy::participant_configuration_writer_↩ publish_mode`

*<<**extension**>> (p. 213)* Publish mode policy for the built-in participant configuration writer.

Determines whether the Discovery built-in participant configuration **DDSDataWriter** (p. 1589) publishes data synchronously or asynchronously and how.

Categories

Immutable (p. 333), **Discovery** (p. 271), **Flow Control** (p. 282)

9.47.2.42 participant_configuration_writer

```
struct DDS_RtpsReliableWriterProtocol_t DDS_DiscoveryConfigQosPolicy::participant_configuration↩
_writer
```

<<***extension***>> *(p. 213)* RTPS protocol-related configuration settings for the RTPS reliable writer associated with a built-in participant configuration writer.

For details, refer to the **DDS_RtpsReliableWriterProtocol_t** (p. 1320)

[default]
low_watermark 0;
high_watermark 1;
heartbeat_period 3.0 seconds;
fast_heartbeat_period 3.0 seconds;
late_joiner_heartbeat_period 3.0 seconds;
virtual_heartbeat_period **DDS_DURATION_INFINITE** (p. 522);
samples_per_virtual_heartbeat **DDS_LENGTH_UNLIMITED** (p. 636);
max_heartbeat_retries 10;
inactivate_nonprogressing_readers **DDS_BOOLEAN_FALSE** (p. 513);
heartbeats_per_max_samples 8;
min_nack_response_delay 0.0 seconds;
max_nack_response_delay 0.0 seconds;
nack_suppression_duration 0.0 seconds;
max_bytes_per_nack_response 131072 bytes;
disable_positive_acks_min_sample_keep_duration 1.0 milliseconds;
disable_positive_acks_max_sample_keep_duration 1.0 seconds;
disable_positive_acks_enable_adaptive_sample_keep_duration **DDS_BOOLEAN_TRUE** (p. 513);
disable_positive_acks_decrease_sample_keep_duration_factor 95;
disable_positive_acks_increase_sample_keep_duration_factor 150;
min_send_window_size **DDS_LENGTH_UNLIMITED** (p. 636);
max_send_window_size **DDS_LENGTH_UNLIMITED** (p. 636);
send_window_update_period 3s;
send_window_increase_factor 105;
send_window_decrease_factor 50;
enable_multicast_periodic_heartbeat **DDS_BOOLEAN_FALSE** (p. 513);
multicast_resend_threshold 2 readers;
disable_repair_piggyback_heartbeat **DDS_BOOLEAN_FALSE** (p. 513);

Categories

Immutable (p. 333), **Discovery** (p. 271), **Reliability** (p. 314)

9.47.2.43 participant_configuration_writer_data_lifecycle

```
struct DDS_WriterDataLifecycleQosPolicy DDS_DiscoveryConfigQosPolicy::participant_configuration↩
_writer_data_lifecycle
```

<<***extension***>> *(p. 213)* Writer data lifecycle settings for a built-in participant configuration writer.

For details, refer to the **DDS_WriterDataLifecycleQosPolicy** (p. 1524). **DDS_WriterDataLifecycleQosPolicy↩ ::autodispose_unregistered_instances** (p. 1525) will always be forced to **DDS_BOOLEAN_TRUE** (p. 513).

Categories

Immutable (p. 333), **Discovery** (p. 271), **Instance Lifecycle** (p. 286)

9.47.2.44 participant_configuration_reader

`struct` **`DDS_RtpsReliableReaderProtocol_t`** `DDS_DiscoveryConfigQosPolicy::participant_configuration_reader`

<<***extension***>> *(p. 213)* RTPS protocol-related configuration settings for the RTPS reliable reader associated with a built-in participant configuration reader.

For details, refer to the **DDS_RtpsReliableReaderProtocol_t** (p. 1314)

[default]
min_heartbeat_response_delay 0.0 seconds;
max_heartbeat_response_delay 0.0 seconds;
heartbeat_suppression_duration 0.0625 seconds;
nack_period 5.0 seconds;
receive_window_size 256;
round_trip_time 0.0 seconds;
app_ack_period 5.0 seconds;
samples_per_app_ack 1;

Categories

Immutable (p. 333), **Discovery** (p. 271), **Reliability** (p. 314)

9.47.2.45 participant_configuration_reader_resource_limits

`struct` **`DDS_BuiltinTopicReaderResourceLimits_t`** `DDS_DiscoveryConfigQosPolicy::participant_configuration_reader_resource_limits`

<<***extension***>> *(p. 213)* Resource limits for the built-in topic participant configuration reader.

For details, see **DDS_BuiltinTopicReaderResourceLimits_t** (p. 815).

Categories

Immutable (p. 333), **Discovery** (p. 271), **Sample Memory Management** (p. 321), **Middleware Memory Management** (p. 297)

9.47.2.46 request_types_filter

```
DDS_String DDS_DiscoveryConfigQosPolicy::request_types_filter
```

*<<**extension**>> (p. 213)* Topic name pattern for requesting all types via Type Lookup Service.

When using TypeObject v2 (enabled via **DDS_DISCOVERYCONFIG_TYPE_LOOKUP_SERVICE_CHANNEL** (p. 592)), this field configures a topic name pattern. Any remote endpoint whose topic name matches this pattern will have all of its types requested by the Type Lookup Service, even if no matching local endpoint exists.

By default, the Type Lookup Service only requests unknown types when a matching local endpoint exists. This setting allows you to proactively retrieve type information for remote endpoints matching specific topic names, which can be useful for applications where types are discovered on the wire (such as those using Dynamic Data).

This is a regex string, so you can use regex syntax to match multiple topic names (including wildcards).

If this is NULL or an empty string, the middleware will only request types for remote endpoints that have a matching local endpoint.

If you are using **DDS_DynamicData** (p. 1013) and relying on discovering types on the wire, you should set this to a value that matches the topic names of your remote endpoints to ensure that you receive type information for those endpoints.

[default] NULL

Categories

Immutable (p. 333), **Discovery** (p. 271)

9.48 DDS_DiscoveryQosPolicy Struct Reference

*<<**extension**>> (p. 213)* Configures the mechanism used by the middleware to automatically discover and connect with new remote applications.

Public Attributes

- struct **DDS_StringSeq enabled_transports**

  *<<**extension**>> (p. 213) The transports available for use by the Discovery mechanism.*
- struct **DDS_StringSeq initial_peers**

  *<<**basic**>> (p. 201) <<**extension**>> (p. 213) Determines the initial list of peers that will be contacted by the Discovery mechanism to send announcements about the presence of this participant.*
- struct **DDS_StringSeq multicast_receive_addresses**

  *<<**basic**>> (p. 201) <<**extension**>> (p. 213) Specifies the multicast group addresses on which discovery-related **meta-traffic** can be received by the DomainParticipant.*
- **DDS_Long metatraffic_transport_priority**

  *<<**extension**>> (p. 213) The transport priority to use for the Discovery meta-traffic.*
- **DDS_Boolean accept_unknown_peers**

  *<<**basic**>> (p. 201) <<**extension**>> (p. 213) Whether to accept a new participant that is not in the initial peers list.*
- **DDS_Boolean enable_endpoint_discovery**

  ***[DEPRECATED]** (p. **384**) <<**extension**>> (p. 213) Whether to automatically enable endpoint discovery for all the remote participants.*

9.48.1 Detailed Description

<<***extension***>> *(p. 213)* Configures the mechanism used by the middleware to automatically discover and connect with new remote applications.

Entity:

DDSDomainParticipant (p. 1619)

Properties:

RxO (p. 556) = N/A
Changeable (p. 556) = **NO** (p. 556)

9.48.2 Usage

This QoS policy identifies where on the network this application can *potentially* discover other applications with which to communicate.

The middleware will periodically send network packets to these locations, announcing itself to any remote applications that may be present, and will listen for announcements from those applications.

This QoS policy is an extension to the DDS standard.

See also

NDDS_DISCOVERY_PEERS (p. 662)

DDS_DiscoveryConfigQosPolicy (p. 934)

9.48.3 Member Data Documentation

9.48.3.1 enabled_transports

`struct DDS_StringSeq DDS_DiscoveryQosPolicy::enabled_transports`

<<***extension***>> *(p. 213)* The transports available for use by the Discovery mechanism.

Only these transports can be used by the discovery mechanism to send meta-traffic via the builtin endpoints (built-in **DDSDataReader** (p. 1556) and **DDSDataWriter** (p. 1589)).

Also determines the unicast addresses on which the Discovery mechanism will listen for meta-traffic. These along with the `domain_id` and `participant_id` determine the unicast locators on which the Discovery mechanism can receive meta-data.

The memory for the strings in this sequence is managed according to the conventions described in **String Conventions** (p. 758). In particular, be careful to avoid a situation in which RTI Connext allocates a string on your behalf and you then reuse that string in such a way that RTI Connext believes it to have more memory allocated to it than it actually does.

Alias names for the builtin transports are defined in **TRANSPORT_BUILTIN** (p. 641). These alias names are case sensitive and should be written in lowercase.

[default] Empty sequence. All the transports available to the DomainParticipant are available for use by the Discovery mechanism.

[range] Sequence of non-null,non-empty strings.

Categories

Immutable (p. 333), **Transports** (p. 327), **Discovery** (p. 271)

9.48.3.2 initial_peers

```
struct DDS_StringSeq DDS_DiscoveryQosPolicy::initial_peers
```

<<***basic***>> *(p. 201)* <<***extension***>> *(p. 213)* Determines the initial list of peers that will be contacted by the Discovery mechanism to send announcements about the presence of this participant.

As part of the participant discovery phase, the **DDSDomainParticipant** (p. 1619) will announce itself to the domain by sending participant DATA messages. The initial_peers specifies the initial list of peers that will be contacted. A remote **DDSDomainParticipant** (p. 1619) is discovered by receiving participant announcements from a remote peer. When the new remote **DDSDomainParticipant** (p. 1619) has been added to the participant's database, the endpoint discovery phase commences and information about the DataWriters and DataReaders is exchanged.

Each element of this list must be a peer descriptor in the proper format (see **Peer Descriptor Format** (p. 663)).

[default] builtin.udpv4://239.255.0.1, builtin.udpv4://127.0.0.1, builtin.shmem:// (See also **NDDS_DISCOVERY_PEERS** (p. 662))

[range] Sequence of arbitrary length.

See also

Peer Descriptor Format (p. 663)

DDSDomainParticipant::add_peer() (p. 1678)

Categories

Immutable (p. 333), **Transports** (p. 327), **Discovery Isolation** (p. 277), **Discovery** (p. 271)

9.48.3.3 multicast_receive_addresses

struct **DDS_StringSeq** DDS_DiscoveryQosPolicy::multicast_receive_addresses

<<***basic***>> *(p. 201)* <<***extension***>> *(p. 213)* Specifies the multicast group addresses on which discovery-related **meta-traffic** can be received by the DomainParticipant.

Each element of this list must be a valid multicast address (IPv4 or IPv6) in the proper format (see **Address Format** (p. 664)).

The `domain_id` determines the multicast port on which the Discovery mechanism can receive meta-data.

If NDDS_DISCOVERY_PEERS does *not* contain a multicast address, then the string sequence **DDS_DiscoveryQos↩ Policy::multicast_receive_addresses** (p. 962) is cleared and the RTI discovery process will not listen for discovery messages via multicast.

If NDDS_DISCOVERY_PEERS contains one or more multicast addresses, the addresses will be stored in **DDS_↩ DiscoveryQosPolicy::multicast_receive_addresses** (p. 962), starting at element 0. They will be stored in the order they appear NDDS_DISCOVERY_PEERS.

Note: Currently, RTI Connext will only listen for discovery traffic on the first multicast address (element 0) in **DDS_↩ DiscoveryQosPolicy::multicast_receive_addresses** (p. 962).

[default] builtin.udpv4://239.255.0.1 (See also **NDDS_DISCOVERY_PEERS** (p. 662))

[range] Sequence of length [0,1], whose elements are multicast addresses. Currently only the first multicast address (if any) is used. The rest are ignored.

See also

Address Format (p. 664)

9.48.3.4 metatraffic_transport_priority

DDS_Long DDS_DiscoveryQosPolicy::metatraffic_transport_priority

<<***extension***>> *(p. 213)* The transport priority to use for the Discovery meta-traffic.

The discovery metatraffic will be sent by the built-in **DDSDataWriter** (p. 1589) using this transport priority.

[default] 0

[range] [0, MAX_UINT]

Categories

Immutable (p. 333), **Transports** (p. 327), **Concurrency** (p. 258), **Discovery** (p. 271)

9.48.3.5 accept_unknown_peers

```
DDS_Boolean DDS_DiscoveryQosPolicy::accept_unknown_peers
```

<<***basic***>> *(p. 201)* <<***extension***>> *(p. 213)* Whether to accept a new participant that is not in the initial peers list.

If **DDS_BOOLEAN_FALSE** (p. 513), the participant will only communicate with those in the initial peers list and those added via **DDSDomainParticipant::add_peer()** (p. 1678).

If **DDS_BOOLEAN_TRUE** (p. 513), the participant will also communicate with all discovered remote participants.

Note: If accept_unknown_peers is **DDS_BOOLEAN_FALSE** (p. 513) and shared memory is disabled, applications on the same node will *not* communicate if only 'localhost' is specified in the peers list. If shared memory is disabled or 'shmem://' is not specified in the peers list, to communicate with other applications on the same node through the loopback interface, you must put the actual node address or hostname in **NDDS_DISCOVERY_PEERS** (p. 662).

[default] DDS_BOOLEAN_TRUE (p. 513)

Categories

Immutable (p. 333), **Transports** (p. 327), **Discovery Isolation** (p. 277), **Discovery** (p. 271)

9.48.3.6 enable_endpoint_discovery

```
DDS_Boolean DDS_DiscoveryQosPolicy::enable_endpoint_discovery
```

[DEPRECATED] (p. 384) <<***extension***>> *(p. 213)* Whether to automatically enable endpoint discovery for all the remote participants.

If **DDS_BOOLEAN_TRUE** (p. 513), endpoint discovery will automatically occur for every discovered remote participant.

If **DDS_BOOLEAN_FALSE** (p. 513), endpoint discovery will be initially disabled and manual activation is required for each discovered participant by calling **DDSDomainParticipant::resume_endpoint_discovery** (p. 1668).

[default] DDS_BOOLEAN_TRUE (p. 513)

Categories

Immutable (p. 333), **Transports** (p. 327), **Discovery Isolation** (p. 277), **Discovery** (p. 271)

9.49 DDS_DomainParticipantConfigParams_t Struct Reference

<<***extension***>> *(p. 432)* Input paramaters for creating a participant from configuration. It allows to modify or override some of the properties of the entities defined in the configuration.

Public Attributes

- int **domain_id**

  *Domain ID from which the **DDSDomainParticipant** (p. 1619) is created.*
- char * **participant_name**

  *Name assigned to the **DDSDomainParticipant** (p. 1619).*
- char * **participant_qos_library_name**

  *QoS library name containing the QoS profile from which the **DDSDomainParticipant** (p. 1619) is created.*
- char * **participant_qos_profile_name**

  *QoS profile name from which the **DDSDomainParticipant** (p. 1619) is created.*
- char * **domain_entity_qos_library_name**

  *QoS library name containing the QoS profile from which the all the **DDSDomainEntity** (p. 1618) defined under the participant configuration are created.*
- char * **domain_entity_qos_profile_name**

  *QoS profile name from which the all the **DDSDomainEntity** (p. 1618) defined under the participant configuration are created.*

9.49.1 Detailed Description

<<***extension***>> *(p. 432)* Input paramaters for creating a participant from configuration. It allows to modify or override some of the properties of the entities defined in the configuration.

9.49.2 Member Data Documentation

9.49.2.1 domain_id

```
int DDS_DomainParticipantConfigParams_t::domain_id
```

Domain ID from which the **DDSDomainParticipant** (p. 1619) is created.

Allows overriding the domain ID defined in the configuration for the participant to be created. If the special value **DDS_DOMAIN_ID_USE_XML_CONFIG** (p. 717) is specified then the ID in the configuration will be used.

9.49.2.2 participant_name

```
char* DDS_DomainParticipantConfigParams_t::participant_name
```

Name assigned to the **DDSDomainParticipant** (p. 1619).

This is the name the name that will be set in the **DDS_DomainParticipantQos::participant_name** (p. 976). It allows overriding the participant name that is generated automatically.

When this member is lexicographically equal to the special value **DDS_ENTITY_NAME_USE_XML_CONFIG** (p. 717) then an automatically generated name will be assigned.

9.49.2.3 participant_qos_library_name

```
char* DDS_DomainParticipantConfigParams_t::participant_qos_library_name
```

QoS library name containing the QoS profile from which the **DDSDomainParticipant** (p. 1619) is created.

Allows overriding the QoS defined in the configuration for the participant to be created. This value only affects to the **DDSDomainParticipant** (p. 1619).

When this member is lexicographically equal to the special value **DDS_QOS_ELEMENT_NAME_USE_XML_CONFIG** (p. 717) then the QoS library from the configuration will be applied. Also, the same action will apply for the QoS profile and the value in **DDS_DomainParticipantConfigParams_t::participant_qos_profile_name** (p. 966) will be ignored.

9.49.2.4 participant_qos_profile_name

```
char* DDS_DomainParticipantConfigParams_t::participant_qos_profile_name
```

QoS profile name from which the **DDSDomainParticipant** (p. 1619) is created.

Allows overriding the QoS defined in the configuration for the participant to be created.This value only affects to the **DDSDomainParticipant** (p. 1619).

When this member is lexicographically equal to the special value **DDS_QOS_ELEMENT_NAME_USE_XML_CONFIG** (p. 717) then the QoS profile from the configuration will be applied. Also, the same action will apply for the QoS library and the value in **DDS_DomainParticipantConfigParams_t::participant_qos_library_name** (p. 965) will be ignored.

9.49.2.5 domain_entity_qos_library_name

```
char* DDS_DomainParticipantConfigParams_t::domain_entity_qos_library_name
```

QoS library name containing the QoS profile from which the all the **DDSDomainEntity** (p. 1618) defined under the participant configuration are created.

Allows overriding the QoS defined in the configuration for the domain entities to be created. This value only affects to the **DDSDomainEntity** (p. 1618).

When this member is lexicographically equal to the special value **DDS_QOS_ELEMENT_NAME_USE_XML_CONFIG** (p. 717) then the QoS library from the configuration will be applied. Also, the same action will apply for the QoS profile and the value in **DDS_DomainParticipantConfigParams_t::domain_entity_qos_profile_name** (p. 966) will be ignored.

9.49.2.6 domain_entity_qos_profile_name

```
char* DDS_DomainParticipantConfigParams_t::domain_entity_qos_profile_name
```

QoS profile name from which the all the **DDSDomainEntity** (p. 1618) defined under the participant configuration are created.

Allows overriding the QoS defined in the configuration for the domain entities to be created. This value only affects to the **DDSDomainEntity** (p. 1618).

When this member is lexicographically equal to the special value **DDS_QOS_ELEMENT_NAME_USE_XML_CONFIG** (p. 717) then the QoS profile from the configuration will be applied. Also, the same action will apply for the QoS library and the value in **DDS_DomainParticipantConfigParams_t::domain_entity_qos_library_name** (p. 966) will be ignored.

9.50 DDS_DomainParticipantFactoryQos Struct Reference

QoS policies supported by a **DDSDomainParticipantFactory** (p. 1694).

Public Member Functions

- bool **operator==** (const **DDS_DomainParticipantFactoryQos** &r) const

  Compares two DomainParticipantFactoryQos objects for equality.
- bool **operator!=** (const **DDS_DomainParticipantFactoryQos** &r) const

  Compares two DomainParticipantFactoryQos objects for inequality.
- **DDS_ReturnCode_t print** () const

  *Prints this **DDS_DomainParticipantFactoryQos** (p. 967) to stdout.*
- **DDS_ReturnCode_t to_string** (char *string, **DDS_UnsignedLong** &string_size) const

  *Obtains a string representation of this **DDS_DomainParticipantQos** (p. 971).*
- **DDS_ReturnCode_t to_string** (char *string, **DDS_UnsignedLong** &string_size, const **DDS_Domain↩ ParticipantFactoryQos** &base) const

  *Obtains a string representation of this **DDS_DomainParticipantFactoryQos** (p. 967).*
- **DDS_ReturnCode_t to_string** (char *string, **DDS_UnsignedLong** &string_size, const **DDS_QosPrintFormat** &format) const

  *Obtains a string representation of this **DDS_DomainParticipantFactoryQos** (p. 967).*
- **DDS_ReturnCode_t to_string** (char *string, **DDS_UnsignedLong** &string_size, const **DDS_Domain↩ ParticipantFactoryQos** &base, const **DDS_QosPrintFormat** &format) const

  *Obtains a string representation of this **DDS_DomainParticipantFactoryQos** (p. 967).*
- **DDS_ReturnCode_t to_string** (char *string, **DDS_UnsignedLong** &string_size, const **DDS_QosPrintAll_t** &) const

  *Obtains a string representation of this **DDS_DomainParticipantFactoryQos** (p. 967).*
- **DDS_ReturnCode_t to_string** (char *string, **DDS_UnsignedLong** &string_size, const **DDS_QosPrintAll_t** &, const **DDS_QosPrintFormat** &format) const

  *Obtains a string representation of this **DDS_DomainParticipantFactoryQos** (p. 967).*

Public Attributes

- struct **DDS_EntityFactoryQosPolicy entity_factory**

  *Entity factory policy, **ENTITY_FACTORY** (p. 600).*
- struct **DDS_SystemResourceLimitsQosPolicy resource_limits**

  *<<**extension**>> (p. 432) System resource limits, **SYSTEM_RESOURCE_LIMITS** (p. 638).*
- struct **DDS_ProfileQosPolicy profile**

  *<<**extension**>> (p. 432) Qos profile policy, **PROFILE** (p. 616).*
- struct **DDS_LoggingQosPolicy logging**

  *<<**extension**>> (p. 432) Logging qos policy, **LOGGING** (p. 609).*
- struct **DDS_MonitoringQosPolicy monitoring**

  *<<**extension**>> (p. 432) Monitoring qos policy, **MONITORING** (p. 610).*

9.50.1 Detailed Description

QoS policies supported by a **DDSDomainParticipantFactory** (p. 1694).

Entity:

DDSDomainParticipantFactory (p. 1694)

See also

All QoS Policies (p. 549) and allowed ranges within each Qos.

9.50.2 Member Function Documentation

9.50.2.1 operator==()

```
bool DDS_DomainParticipantFactoryQos::operator== (
            const  DDS_DomainParticipantFactoryQos & r ) const  [inline]
```

Compares two DomainParticipantFactoryQos objects for equality.

See also

DDS_DomainParticipantFactoryQos_equals (p. 44)

References **DDS_DomainParticipantFactoryQos_equals()**.

9.50.2.2 operator"!=()

```
bool DDS_DomainParticipantFactoryQos::operator!= (
            const  DDS_DomainParticipantFactoryQos & r ) const  [inline]
```

Compares two DomainParticipantFactoryQos objects for inequality.

See also

DDS_DomainParticipantFactoryQos_equals (p. 44)

References **DDS_DomainParticipantFactoryQos_equals()**.

9.50.3 Member Data Documentation

9.50.3.1 entity_factory

struct **DDS_EntityFactoryQosPolicy** DDS_DomainParticipantFactoryQos::entity_factory

Entity factory policy, **ENTITY_FACTORY** (p. 600).

9.50.3.2 resource_limits

struct **DDS_SystemResourceLimitsQosPolicy** DDS_DomainParticipantFactoryQos::resource_limits

<<***extension***>> *(p. 432)* System resource limits, **SYSTEM_RESOURCE_LIMITS** (p. 638).

Note: This QoS policy cannot be configured from XML configuration files.

9.50.3.3 profile

struct **DDS_ProfileQosPolicy** DDS_DomainParticipantFactoryQos::profile

<<***extension***>> *(p. 432)* Qos profile policy, **PROFILE** (p. 616).

Note: This QoS policy cannot be configured from XML configuration files.

9.50.3.4 logging

struct **DDS_LoggingQosPolicy** DDS_DomainParticipantFactoryQos::logging

<<***extension***>> *(p. 432)* Logging qos policy, **LOGGING** (p. 609).

9.50.3.5 monitoring

struct **DDS_MonitoringQosPolicy** DDS_DomainParticipantFactoryQos::monitoring

<<***extension***>> *(p. 432)* Monitoring qos policy, **MONITORING** (p. 610).

This QoS policy is used to collect and emit telemetry data of a Connext application using RTI Monitoring Library 2.0.

9.51 DDS_DomainParticipantProtocolStatus Struct Reference

<<***extension***>> *(*p. *432)* The status of a participant's protocol related metrics, like the number of corrupted messages, change in corrupted messages and timestamp of the last corrupted message.

Public Attributes

- **DDS_LongLong corrupted_rtps_message_count**

  The number of corrupted RTPS messages detected by the domain participant.
- **DDS_LongLong corrupted_rtps_message_count_change**

  The incremental change in the number of corrupted RTPS messages detected by the domain participant since the last time the status was read.
- struct **DDS_Time_t last_corrupted_message_timestamp**

  The timestamp when the last corrupted RTPS message was detected by the domain participant.

9.51.1 Detailed Description

<<***extension***>> *(*p. *432)* The status of a participant's protocol related metrics, like the number of corrupted messages, change in corrupted messages and timestamp of the last corrupted message.

Entity:

DDSDomainParticipant (p. 1619) The corrupted messages are detected by validating the received CRC. The participant protocol status can be obtained by enabling **DDS_WireProtocolQosPolicy::compute_crc** (p. 1518) and **DDS_WireProtocolQosPolicy::check_crc** (p. 1518) at the publishing and subscribing application respectively.

9.51.2 Member Data Documentation

9.51.2.1 corrupted_rtps_message_count

```
DDS_LongLong DDS_DomainParticipantProtocolStatus::corrupted_rtps_message_count
```

The number of corrupted RTPS messages detected by the domain participant.

Counts the corrupted RTPS messages received by the participant. It includes messages belonging to discovery and user traffic.

9.51.2.2 corrupted_rtps_message_count_change

```
DDS_LongLong DDS_DomainParticipantProtocolStatus::corrupted_rtps_message_count_change
```

The incremental change in the number of corrupted RTPS messages detected by the domain participant since the last time the status was read.

9.51.2.3 last_corrupted_message_timestamp

```
struct DDS_Time_t DDS_DomainParticipantProtocolStatus::last_corrupted_message_timestamp
```

The timestamp when the last corrupted RTPS message was detected by the domain participant.

9.52 DDS_DomainParticipantQos Struct Reference

QoS policies supported by a **DDSDomainParticipant** (p. 1619) entity.

Public Member Functions

- bool **operator==** (const **DDS_DomainParticipantQos** &r) const

  Compares two DomainParticipantQos objects for equality.
- bool **operator!=** (const **DDS_DomainParticipantQos** &r) const

  Compares two DomainParticipantQos objects for inequality.
- **DDS_ReturnCode_t print** () const

  *Prints this **DDS_DomainParticipantQos** (p. 971) to stdout.*
- **DDS_ReturnCode_t to_string** (char *string, **DDS_UnsignedLong** &string_size) const

  *Obtains a string representation of this **DDS_DomainParticipantQos** (p. 971).*
- **DDS_ReturnCode_t to_string** (char *string, **DDS_UnsignedLong** &string_size, const **DDS_DomainParticipantQos** &base) const

  *Obtains a string representation of this **DDS_DomainParticipantQos** (p. 971).*
- **DDS_ReturnCode_t to_string** (char *string, **DDS_UnsignedLong** &string_size, const **DDS_QosPrintFormat** &format) const

  *Obtains a string representation of this **DDS_DomainParticipantQos** (p. 971).*
- **DDS_ReturnCode_t to_string** (char *string, **DDS_UnsignedLong** &string_size, const **DDS_DomainParticipantQos** &base, const **DDS_QosPrintFormat** &format) const

  *Obtains a string representation of this **DDS_DomainParticipantQos** (p. 971).*
- **DDS_ReturnCode_t to_string** (char *string, **DDS_UnsignedLong** &string_size, const **DDS_QosPrintAll_t** &) const

  *Obtains a string representation of this **DDS_DomainParticipantQos** (p. 971).*
- **DDS_ReturnCode_t to_string** (char *string, **DDS_UnsignedLong** &string_size, const **DDS_QosPrintAll_t** &, const **DDS_QosPrintFormat** &format) const

  *Obtains a string representation of this **DDS_DomainParticipantQos** (p. 971).*

Public Attributes

- struct **DDS_UserDataQosPolicy user_data**

  *User data policy, **USER_DATA** (p. 653).*
- struct **DDS_EntityFactoryQosPolicy entity_factory**

  *<<**basic**>> (p. 201) Entity factory policy, **ENTITY_FACTORY** (p. 600).*
- struct **DDS_WireProtocolQosPolicy wire_protocol**

  *<<**extension**>> (p. 432) Wire Protocol policy, **WIRE_PROTOCOL** (p. 654).*
- struct **DDS_TransportBuiltinQosPolicy transport_builtin**

*<<**extension**>> (p. 432) Transport Builtin policy, **TRANSPORT_BUILTIN** (p. 641).*

- struct **DDS_TransportUnicastQosPolicy default_unicast**

  *<<**extension**>> (p. 432) Default Unicast Transport policy, **TRANSPORT_UNICAST** (p. 649).*
- struct **DDS_DiscoveryQosPolicy discovery**

  *<<**extension**>> (p. 432) Discovery policy, **DISCOVERY** (p. 586).*
- struct **DDS_DomainParticipantResourceLimitsQosPolicy resource_limits**

  *<<**extension**>> (p. 432) Domain participant resource limits policy, **DOMAIN_PARTICIPANT_RESOURCE_LIMITS** (p. 594).*
- struct **DDS_EventQosPolicy event**

  *<<**extension**>> (p. 432) Event policy, **EVENT** (p. 602).*
- struct **DDS_ReceiverPoolQosPolicy receiver_pool**

  *<<**extension**>> (p. 432) Receiver pool policy, **RECEIVER_POOL** (p. 630).*
- struct **DDS_DatabaseQosPolicy database**

  *<<**extension**>> (p. 432) Database policy, **DATABASE** (p. 563).*
- struct **DDS_DiscoveryConfigQosPolicy discovery_config**

  *<<**extension**>> (p. 432) Discovery config policy, **DISCOVERY_CONFIG** (p. 587).*
- struct **DDS_PropertyQosPolicy property**

  *<<**basic**>> (p. 201) <<**extension**>> (p. 432) Property policy, **PROPERTY** (p. 617). See also `Property Reference Guide`.*
- struct **DDS_EntityNameQosPolicy participant_name**

  *<<**basic**>> (p. 201) <<**extension**>> (p. 432) The participant name. **ENTITY_NAME** (p. 601)*
- struct **DDS_TransportMulticastMappingQosPolicy multicast_mapping**

  *<<**extension**>> (p. 432) The multicast mapping policy. **TRANSPORT_MULTICAST_MAPPING** (p. 646)*
- struct **DDS_ServiceQosPolicy service**

  *<<**extension**>> (p. 432) The service qos policy. **SERVICE** (p. 636)*
- struct **DDS_PartitionQosPolicy partition**

  *<<**basic**>> (p. 201) <<**extension**>> (p. 432) The partition qos policy. **PARTITION** (p. 614)*
- struct **DDS_TypeSupportQosPolicy type_support**

  *<<**extension**>> (p. 432) Type support data, **TYPESUPPORT** (p. 651).*

9.52.1 Detailed Description

QoS policies supported by a **DDSDomainParticipant** (p. 1619) entity.

Certain members must be set in a consistent manner:

Length of **DDS_DomainParticipantQos::user_data** (p. 973) .value <= **DDS_DomainParticipantQos::resource_↩ limits** (p. 974) .participant_user_data_max_length

For **DDS_DomainParticipantQos::discovery_config** (p. 975) .publication_writer high_watermark <= **DDS_DomainParticipantQos::resource_limits** (p. 974) .local_writer_allocation .max_count heartbeats_per_max_samples <= **DDS_DomainParticipantQos::resource_limits** (p. 974) .local_writer_allocation.↩ max_count

For **DDS_DomainParticipantQos::discovery_config** (p. 975) .suscription_writer high_watermark <= **DDS_DomainParticipantQos::resource_limits** (p. 974) .local_reader_allocation.max_count heartbeats_per_max_samples <= **DDS_DomainParticipantQos::resource_limits** (p. 974) .local_reader_allocation.↩ max_count

If any of the above are not true, **DDSDomainParticipant::set_qos** (p. 1677) and **DDSDomainParticipant::set↩ _qos_with_profile** (p. 1677) and **DDSDomainParticipantFactory::set_default_participant_qos** (p. 1698) will fail with **DDS_RETCODE_INCONSISTENT_POLICY** (p. 533), and **DDSDomainParticipantFactory::create_participant** (p. 1710) will fail.

Entity:

DDSDomainParticipant (p. 1619)

See also

All QoS Policies (p. 549) and allowed ranges within each Qos.

NDDS_DISCOVERY_PEERS (p. 662)

9.52.2 Member Function Documentation

9.52.2.1 operator==()

```
bool DDS_DomainParticipantQos::operator== (
            const  DDS_DomainParticipantQos & r ) const  [inline]
```

Compares two DomainParticipantQos objects for equality.

See also

DDS_DomainParticipantQos_equals (p. 53)

References **DDS_DomainParticipantQos_equals()**.

9.52.2.2 operator"!=()

```
bool DDS_DomainParticipantQos::operator!= (
            const  DDS_DomainParticipantQos & r ) const  [inline]
```

Compares two DomainParticipantQos objects for inequality.

See also

DDS_DomainParticipantQos_equals (p. 53)

References **DDS_DomainParticipantQos_equals()**.

9.52.3 Member Data Documentation

9.52.3.1 user_data

`struct` **DDS_UserDataQosPolicy** DDS_DomainParticipantQos::user_data

User data policy, **USER_DATA** (p. 653).

9.52.3.2 entity_factory

`struct` **DDS_EntityFactoryQosPolicy** DDS_DomainParticipantQos::entity_factory

<<***basic***>> *(p. 201)* Entity factory policy, **ENTITY_FACTORY** (p. 600).

9.52.3.3 wire_protocol

`struct` **DDS_WireProtocolQosPolicy** DDS_DomainParticipantQos::wire_protocol

<<***extension***>> *(p. 432)* Wire Protocol policy, **WIRE_PROTOCOL** (p. 654).

The wire protocol (RTPS) attributes associated with the participant.

9.52.3.4 transport_builtin

`struct` **DDS_TransportBuiltinQosPolicy** DDS_DomainParticipantQos::transport_builtin

<<***extension***>> *(p. 432)* Transport Builtin policy, **TRANSPORT_BUILTIN** (p. 641).

9.52.3.5 default_unicast

`struct` **DDS_TransportUnicastQosPolicy** DDS_DomainParticipantQos::default_unicast

<<***extension***>> *(p. 432)* Default Unicast Transport policy, **TRANSPORT_UNICAST** (p. 649).

9.52.3.6 discovery

`struct` **DDS_DiscoveryQosPolicy** DDS_DomainParticipantQos::discovery

<<***extension***>> *(p. 432)* Discovery policy, **DISCOVERY** (p. 586).

9.52.3.7 resource_limits

`struct` **DDS_DomainParticipantResourceLimitsQosPolicy** `DDS_DomainParticipantQos::resource_limits`

<<***extension***>> *(p. 432)* Domain participant resource limits policy, **DOMAIN_PARTICIPANT_RESOURCE_LIMITS** (p. 594).

9.52.3.8 event

`struct` **DDS_EventQosPolicy** `DDS_DomainParticipantQos::event`

<<***extension***>> *(p. 432)* Event policy, **EVENT** (p. 602).

9.52.3.9 receiver_pool

`struct` **DDS_ReceiverPoolQosPolicy** `DDS_DomainParticipantQos::receiver_pool`

<<***extension***>> *(p. 432)* Receiver pool policy, **RECEIVER_POOL** (p. 630).

9.52.3.10 database

`struct` **DDS_DatabaseQosPolicy** `DDS_DomainParticipantQos::database`

<<***extension***>> *(p. 432)* Database policy, **DATABASE** (p. 563).

9.52.3.11 discovery_config

`struct` **DDS_DiscoveryConfigQosPolicy** `DDS_DomainParticipantQos::discovery_config`

<<***extension***>> *(p. 432)* Discovery config policy, **DISCOVERY_CONFIG** (p. 587).

9.52.3.12 property

`struct` **DDS_PropertyQosPolicy** DDS_DomainParticipantQos::property

*<<**basic**>> (p. 201) <<**extension**>> (p. 432)* Property policy, **PROPERTY** (p. 617). See also `Property Reference Guide`.

9.52.3.13 participant_name

`struct` **DDS_EntityNameQosPolicy** DDS_DomainParticipantQos::participant_name

*<<**basic**>> (p. 201) <<**extension**>> (p. 432)* The participant name. **ENTITY_NAME** (p. 601)

9.52.3.14 multicast_mapping

`struct` **DDS_TransportMulticastMappingQosPolicy** DDS_DomainParticipantQos::multicast_mapping

*<<**extension**>> (p. 432)* The multicast mapping policy. **TRANSPORT_MULTICAST_MAPPING** (p. 646)

9.52.3.15 service

`struct` **DDS_ServiceQosPolicy** DDS_DomainParticipantQos::service

*<<**extension**>> (p. 432)* The service qos policy. **SERVICE** (p. 636)

9.52.3.16 partition

`struct` **DDS_PartitionQosPolicy** DDS_DomainParticipantQos::partition

*<<**basic**>> (p. 201) <<**extension**>> (p. 432)* The partition qos policy. **PARTITION** (p. 614)

9.52.3.17 type_support

`struct` **DDS_TypeSupportQosPolicy** DDS_DomainParticipantQos::type_support

*<<**extension**>> (p. 432)* Type support data, **TYPESUPPORT** (p. 651).

Optional value that is passed to a type plugin's on_participant_attached function.

9.53 DDS_DomainParticipantResourceLimitsQosPolicy Struct Reference

<<***extension***>> *(p. 213)* Various settings that configure how a **DDSDomainParticipant** (p. 1619) allocates and uses physical memory for internal resources, including the maximum sizes of various properties.

Public Attributes

- struct **DDS_AllocationSettings_t local_writer_allocation**

  <<***extension***>> *(p. 213) Allocation settings applied to local DataWriters.*
- struct **DDS_AllocationSettings_t local_reader_allocation**

  <<***extension***>> *(p. 213) Allocation settings applied to local DataReaders.*
- struct **DDS_AllocationSettings_t local_publisher_allocation**

  <<***extension***>> *(p. 213) Allocation settings applied to local Publisher.*
- struct **DDS_AllocationSettings_t local_subscriber_allocation**

  <<***extension***>> *(p. 213) Allocation settings applied to local Subscriber.*
- struct **DDS_AllocationSettings_t local_topic_allocation**

  <<***extension***>> *(p. 213) Allocation settings applied to local Topic.*
- struct **DDS_AllocationSettings_t remote_writer_allocation**

  <<***extension***>> *(p. 213) Allocation settings applied to remote DataWriters.*
- struct **DDS_AllocationSettings_t remote_reader_allocation**

  <<***extension***>> *(p. 213) Allocation settings applied to remote DataReaders.*
- struct **DDS_AllocationSettings_t remote_participant_allocation**

  <<***extension***>> *(p. 213) Allocation settings applied to remote DomainParticipants.*
- struct **DDS_AllocationSettings_t matching_writer_reader_pair_allocation**

  <<***extension***>> *(p. 213) Allocation settings applied to matching local writer and remote/local reader pairs.*
- struct **DDS_AllocationSettings_t matching_reader_writer_pair_allocation**

  <<***extension***>> *(p. 213) Allocation settings applied to matching local reader and remote/local writer pairs.*
- struct **DDS_AllocationSettings_t ignored_entity_allocation**

  <<***extension***>> *(p. 213) Allocation settings applied to ignored entities.*
- struct **DDS_AllocationSettings_t content_filtered_topic_allocation**

  <<***extension***>> *(p. 213) Allocation settings applied to content filtered topic.*
- struct **DDS_AllocationSettings_t content_filter_allocation**

  <<***extension***>> *(p. 213) Allocation settings applied to content filter.*
- struct **DDS_AllocationSettings_t read_condition_allocation**

  <<***extension***>> *(p. 213) Allocation settings applied to read condition pool.*
- struct **DDS_AllocationSettings_t query_condition_allocation**

  <<***extension***>> *(p. 213) Allocation settings applied to query condition pool.*
- struct **DDS_AllocationSettings_t outstanding_asynchronous_sample_allocation**

  <<***extension***>> *(p. 213) Allocation settings applied to the maximum number of samples (from all **DDSDataWriter** (p. 1589)) waiting to be asynchronously written.*
- struct **DDS_AllocationSettings_t flow_controller_allocation**

  <<***extension***>> *(p. 213) Allocation settings applied to flow controllers.*
- **DDS_Long local_writer_hash_buckets**

  [DEPRECATED] *(p. **384**)* <<***extension***>> *(p. 213) Hash_Buckets settings applied to local DataWriters.*
- **DDS_Long local_reader_hash_buckets**

  [DEPRECATED] *(p. **384**)* <<***extension***>> *(p. 213) Number of hash buckets for local DataReaders.*

- **DDS_Long local_publisher_hash_buckets**

  [DEPRECATED] (p. 384) <<extension>> (p. 213) Number of hash buckets for local Publisher.
- **DDS_Long local_subscriber_hash_buckets**

  [DEPRECATED] (p. 384) <<extension>> (p. 213) Number of hash buckets for local Subscriber.
- **DDS_Long local_topic_hash_buckets**

  [DEPRECATED] (p. 384) <<extension>> (p. 213) Number of hash buckets for local Topic.
- **DDS_Long remote_writer_hash_buckets**

  [DEPRECATED] (p. 384) <<extension>> (p. 213) Number of hash buckets for remote DataWriters.
- **DDS_Long remote_reader_hash_buckets**

  [DEPRECATED] (p. 384) <<extension>> (p. 213) Number of hash buckets for remote DataReaders.
- **DDS_Long remote_participant_hash_buckets**

  [DEPRECATED] (p. 384) <<extension>> (p. 213) Number of hash buckets for remote DomainParticipants.
- **DDS_Long matching_writer_reader_pair_hash_buckets**

  [DEPRECATED] (p. 384) <<extension>> (p. 213) Number of hash buckets for matching local writer and remote/local reader pairs.
- **DDS_Long matching_reader_writer_pair_hash_buckets**

  [DEPRECATED] (p. 384) <<extension>> (p. 213) Number of hash buckets for matching local reader and remote/local writer pairs.
- **DDS_Long ignored_entity_hash_buckets**

  [DEPRECATED] (p. 384) <<extension>> (p. 213) Number of hash buckets for ignored entities.
- **DDS_Long content_filtered_topic_hash_buckets**

  [DEPRECATED] (p. 384) <<extension>> (p. 213) Number of hash buckets for content filtered topics.
- **DDS_Long content_filter_hash_buckets**

  [DEPRECATED] (p. 384) <<extension>> (p. 213) Number of hash buckets for content filters.
- **DDS_Long flow_controller_hash_buckets**

  [DEPRECATED] (p. 384) <<extension>> (p. 213) Number of hash buckets for flow controllers.
- **DDS_Long max_gather_destinations**

  <<extension>> (p. 213) Maximum number of destinations per RTI Connext send.
- **DDS_Long participant_user_data_max_length**

  *<<extension>> (p. 213) Maximum length of user data in **DDS_DomainParticipantQos** (p. 971) and **DDS_ParticipantBuiltinTopicData** (p. 1229)*
- **DDS_Long topic_data_max_length**

  *<<extension>> (p. 213) Maximum length of topic data in **DDS_TopicQos** (p. 1397), **DDS_TopicBuiltinTopicData** (p. 1391), **DDS_PublicationBuiltinTopicData** (p. 1264) and **DDS_SubscriptionBuiltinTopicData** (p. 1370)*
- **DDS_Long publisher_group_data_max_length**

  *<<extension>> (p. 213) Maximum length of group data in **DDS_PublisherQos** (p. 1276) and **DDS_PublicationBuiltinTopicData** (p. 1264)*
- **DDS_Long subscriber_group_data_max_length**

  *<<extension>> (p. 213) Maximum length of group data in **DDS_SubscriberQos** (p. 1367) and **DDS_SubscriptionBuiltinTopicData** (p. 1370)*
- **DDS_Long writer_user_data_max_length**

  *<<extension>> (p. 213) Maximum length of user data in **DDS_DataWriterQos** (p. 909) and **DDS_PublicationBuiltinTopicData** (p. 1264)*
- **DDS_Long reader_user_data_max_length**

  *<<extension>> (p. 213) Maximum length of user data in **DDS_DataReaderQos** (p. 858) and **DDS_SubscriptionBuiltinTopicData** (p. 1370)*
- **DDS_Long max_partitions**

  *<<extension>> (p. 213) Maximum number of partition name strings allowable in a **DDS_PartitionQosPolicy** (p. 1240).*
- **DDS_Long max_partition_cumulative_characters**

*<<**extension**>> (p. 213) Maximum number of combined characters allowable in all partition names in a **DDS_PartitionQosPolicy** (p. 1240).*

- **DDS_Long type_code_max_serialized_length**

  ***[DEPRECATED] (**p. **384)** <<**extension**>> (p. 213) Maximum size of serialized string for type code.*

- **DDS_Long type_object_max_serialized_length**

  *<<**extension**>> (p. 213) The maximum length, in bytes, that the buffer to serialize a TypeObject v1 can consume.*

- **DDS_Long serialized_type_object_dynamic_allocation_threshold**

  *<<**extension**>> (p. 213) A threshold, in bytes, for dynamic memory allocation for the serialized TypeObject v1.*

- **DDS_Long type_object_max_deserialized_length**

  *<<**extension**>> (p. 213) The maximum number of bytes that a deserialized TypeObject v1 can consume.*

- **DDS_Long deserialized_type_object_dynamic_allocation_threshold**

  *<<**extension**>> (p. 213) A threshold, in bytes, for dynamic memory allocation for the deserialized TypeObject v1.*

- **DDS_Long contentfilter_property_max_length**

  *<<**extension**>> (p. 213) This field is the maximum length of all data related to a Content-filtered topic.*

- **DDS_Long channel_seq_max_length**

  *<<**extension**>> (p. 213) Maximum number of channels that can be specified in **DDS_MultiChannelQosPolicy** (p. 1214) for MultiChannel DataWriters.*

- **DDS_Long channel_filter_expression_max_length**

  *<<**extension**>> (p. 213) Maximum length of a channel **DDS_ChannelSettings_t::filter_expression** (p. 822) in a MultiChannel DataWriter.*

- **DDS_Long participant_property_list_max_length**

  *<<**extension**>> (p. 213) Maximum number of properties associated with the **DDSDomainParticipant** (p. 1619).*

- **DDS_Long participant_property_string_max_length**

  *<<**extension**>> (p. 213) Maximum string length of the properties associated with the **DDSDomainParticipant** (p. 1619).*

- **DDS_Long writer_property_list_max_length**

  *<<**extension**>> (p. 213) Maximum number of properties associated with a **DDSDataWriter** (p. 1589).*

- **DDS_Long writer_property_string_max_length**

  *<<**extension**>> (p. 213) Maximum string length of the properties associated with a **DDSDataWriter** (p. 1589).*

- **DDS_Long reader_property_list_max_length**

  *<<**extension**>> (p. 213) Maximum number of properties associated with a **DDSDataReader** (p. 1556).*

- **DDS_Long reader_property_string_max_length**

  *<<**extension**>> (p. 213) Maximum string length of the properties associated with a **DDSDataReader** (p. 1556).*

- **DDS_Long max_endpoint_groups**

  *<<**extension**>> (p. 213) Maximum number of **DDS_EndpointGroup_t** (p. 1142) allowable in a **DDS_AvailabilityQosPolicy** (p. 805).*

- **DDS_Long max_endpoint_group_cumulative_characters**

  *<<**extension**>> (p. 213) Maximum number of combined role_name characters allowed in all **DDS_EndpointGroup_t** (p. 1142) in a **DDS_AvailabilityQosPolicy** (p. 805).*

- **DDS_Long transport_info_list_max_length**

  *<<**extension**>> (p. 213) Maximum number of installed transports to send and receive information about in **DDS_ParticipantBuiltinTopicData::transport_info** (p. 1232).*

- **DDS_DomainParticipantResourceLimitsIgnoredEntityReplacementKind ignored_entity_replacement_kind**

  ***[DEPRECATED] (**p. **384)** <<**extension**>> (p. 213) Replacement policy for the ignored entities. It sets what entity can be replaced when resource limits set in **DDS_DomainParticipantResourceLimitsQosPolicy::ignored_entity_allocation** (p. 984) are reached.*

- struct **DDS_AllocationSettings_t remote_topic_query_allocation**

  *<<**extension**>> (p. 213) Allocation settings applied to remote TopicQueries.*

- **DDS_Long remote_topic_query_hash_buckets**

  [DEPRECATED] (p. ***384***) *<<**extension**>> (p. 213) Number of hash buckets for remote TopicQueries.*
- **DDS_Long writer_data_tag_list_max_length**

  *<<**extension**>> (p. 213) Maximum number of data tags associated with a **DDSDataWriter*** (p. *1589).*
- **DDS_Long writer_data_tag_string_max_length**

  *<<**extension**>> (p. 213) Maximum string length of the data tags associated with a **DDSDataWriter*** (p. *1589).*
- **DDS_Long reader_data_tag_list_max_length**

  *<<**extension**>> (p. 213) Maximum number of data tags associated with a **DDSDataReader*** (p. *1556).*
- **DDS_Long reader_data_tag_string_max_length**

  *<<**extension**>> (p. 213) Maximum string length of the data tags associated with a **DDSDataReader*** (p. *1556).*
- **DDS_UnsignedLong shmem_ref_transfer_mode_max_segments**

  *<<**extension**>> (p. 213) Maximum number of segments created by all DataWriters belonging to a **DDSDomain**↩**Participant*** (p. *1619).*

9.53.1 Detailed Description

*<<**extension**>> (p. 213)* Various settings that configure how a **DDSDomainParticipant** (p. 1619) allocates and uses physical memory for internal resources, including the maximum sizes of various properties.

This QoS policy sets maximum size limits on variable-length parameters used by the participant and its contained Entities. It also controls the initial and maximum sizes of data structures used by the participant to store information about locally-created and remotely-discovered entities (such as DataWriters/DataReaders), as well as parameters used by the internal database to size the hash tables it uses.

By default, a **DDSDomainParticipant** (p. 1619) is allowed to dynamically allocate memory as needed as users create local Entities such as **DDSDataWriter** (p. 1589) and **DDSDataReader** (p. 1556) or as the participant discovers new applications to store their information. By setting fixed values for the maximum parameters in this QosPolicy, you can bound the memory that can be allocated by a DomainParticipant. In addition, by setting the initial values to the maximum values, you can reduce the amount of memory allocated by DomainParticipants after the initialization period. Notice that memory can still be allocated dynamically after the initialization period. For example, when a new local **DDSDataWriter** (p. 1589) or **DDSDataReader** (p. 1556) is created, the initial memory required for its queue is allocated dynamically.

The maximum sizes of different variable-length parameters such as the number of partitions that can be stored in the **DDS_PartitionQosPolicy** (p. 1240), the maximum length of data store in the **DDS_UserDataQosPolicy** (p. 1504) and **DDS_GroupDataQosPolicy** (p. 1160), and many others can be changed from their defaults using this QoS policy. However, it is important that all DomainParticipants that need to communicate with each other use the *same set* of maximum values. Otherwise, when these parameters are propagated from one **DDSDomainParticipant** (p. 1619) to another, a **DDSDomainParticipant** (p. 1619) with a smaller maximum length may reject the parameter, resulting in an error.

An important parameter in this QoS policy that is often changed by users is **DDS_DomainParticipantResource**↩**LimitsQosPolicy::type_object_max_serialized_length** (p. 994).

This QoS policy is an extension to the DDS standard.

Entity:

DDSDomainParticipant (p. 1619)

Properties:

RxO (p. 556) = N/A
Changeable (p. 556) = **NO** (p. 556)

9.53.2 Member Data Documentation

9.53.2.1 local_writer_allocation

`struct DDS_AllocationSettings_t DDS_DomainParticipantResourceLimitsQosPolicy::local_writer_allocation`

<<***extension***>> *(p. 213)* Allocation settings applied to local DataWriters.

[default] initial_count = 16; max_count = **DDS_LENGTH_UNLIMITED** (p. 636); incremental_count = -1

[range] See allowed ranges in struct **DDS_AllocationSettings_t** (p. 796)

Categories

Immutable (p. 333), **Middleware Memory Management** (p. 297)

9.53.2.2 local_reader_allocation

`struct DDS_AllocationSettings_t DDS_DomainParticipantResourceLimitsQosPolicy::local_reader_allocation`

<<***extension***>> *(p. 213)* Allocation settings applied to local DataReaders.

[default] initial_count = 16; max_count = **DDS_LENGTH_UNLIMITED** (p. 636); incremental_count = -1

[range] See allowed ranges in struct **DDS_AllocationSettings_t** (p. 796)

Categories

Immutable (p. 333), **Middleware Memory Management** (p. 297)

9.53.2.3 local_publisher_allocation

`struct DDS_AllocationSettings_t DDS_DomainParticipantResourceLimitsQosPolicy::local_publisher_allocation`

<<***extension***>> *(p. 213)* Allocation settings applied to local Publisher.

[default] initial_count = 4; max_count = **DDS_LENGTH_UNLIMITED** (p. 636); incremental_count = -1

[range] See allowed ranges in struct **DDS_AllocationSettings_t** (p. 796)

Categories

Immutable (p. 333), **Middleware Memory Management** (p. 297)

9.53.2.4 local_subscriber_allocation

`struct` **DDS_AllocationSettings_t** `DDS_DomainParticipantResourceLimitsQosPolicy::local_subscriber_allocation`

<<***extension***>> *(p. 213)* Allocation settings applied to local Subscriber.

[default] initial_count = 4; max_count = **DDS_LENGTH_UNLIMITED** (p. 636); incremental_count = -1

[range] See allowed ranges in struct **DDS_AllocationSettings_t** (p. 796)

Categories

Immutable (p. 333), **Middleware Memory Management** (p. 297)

9.53.2.5 local_topic_allocation

`struct` **DDS_AllocationSettings_t** `DDS_DomainParticipantResourceLimitsQosPolicy::local_topic_allocation`

<<***extension***>> *(p. 213)* Allocation settings applied to local Topic.

[default] initial_count = 16; max_count = **DDS_LENGTH_UNLIMITED** (p. 636); incremental_count = -1

[range] See allowed ranges in struct **DDS_AllocationSettings_t** (p. 796)

Categories

Immutable (p. 333), **Middleware Memory Management** (p. 297)

9.53.2.6 remote_writer_allocation

`struct` **DDS_AllocationSettings_t** `DDS_DomainParticipantResourceLimitsQosPolicy::remote_writer_allocation`

<<***extension***>> *(p. 213)* Allocation settings applied to remote DataWriters.

Remote DataWriters include all DataWriters, both local and remote.

[default] initial_count = 64; max_count = **DDS_LENGTH_UNLIMITED** (p. 636); incremental_count = -1

[range] See allowed ranges in struct **DDS_AllocationSettings_t** (p. 796)

Categories

Immutable (p. 333), **Middleware Memory Management** (p. 297)

9.53.2.7 remote_reader_allocation

struct **DDS_AllocationSettings_t** DDS_DomainParticipantResourceLimitsQosPolicy::remote_reader_allocation

<<***extension***>> *(p. 213)* Allocation settings applied to remote DataReaders.

Remote DataReaders include all DataReaders, both local and remote.

[default] initial_count = 64; max_count = **DDS_LENGTH_UNLIMITED** (p. 636); incremental_count = -1

[range] See allowed ranges in struct **DDS_AllocationSettings_t** (p. 796)

Categories

Immutable (p. 333), **Middleware Memory Management** (p. 297)

9.53.2.8 remote_participant_allocation

struct **DDS_AllocationSettings_t** DDS_DomainParticipantResourceLimitsQosPolicy::remote_participant_allocation

<<***extension***>> *(p. 213)* Allocation settings applied to remote DomainParticipants.

Remote DomainParticipants include all DomainParticipants, both local and remote.

[default] initial_count = 16; max_count = **DDS_LENGTH_UNLIMITED** (p. 636); incremental_count = -1

[range] See allowed ranges in struct **DDS_AllocationSettings_t** (p. 796)

Categories

Immutable (p. 333), **Middleware Memory Management** (p. 297)

9.53.2.9 matching_writer_reader_pair_allocation

struct **DDS_AllocationSettings_t** DDS_DomainParticipantResourceLimitsQosPolicy::matching_writer_reader_pair_allocation

<<***extension***>> *(p. 213)* Allocation settings applied to matching local writer and remote/local reader pairs.

[default] initial_count = 32; max_count = **DDS_LENGTH_UNLIMITED** (p. 636); incremental_count = -1

[range] See allowed ranges in struct **DDS_AllocationSettings_t** (p. 796)

Categories

Immutable (p. 333), **Middleware Memory Management** (p. 297)

9.53.2.10 matching_reader_writer_pair_allocation

`struct` **`DDS_AllocationSettings_t`** `DDS_DomainParticipantResourceLimitsQosPolicy::matching_reader_↩ writer_pair_allocation`

<<***extension***>> *(p. 213)* Allocation settings applied to matching local reader and remote/local writer pairs.

[default] initial_count = 32; max_count = **DDS_LENGTH_UNLIMITED** (p. 636); incremental_count = -1

[range] See allowed ranges in struct **DDS_AllocationSettings_t** (p. 796)

Categories

Immutable (p. 333), **Middleware Memory Management** (p. 297)

9.53.2.11 ignored_entity_allocation

`struct` **`DDS_AllocationSettings_t`** `DDS_DomainParticipantResourceLimitsQosPolicy::ignored_entity_↩ allocation`

<<***extension***>> *(p. 213)* Allocation settings applied to ignored entities.

[default] initial_count = 8; max_count = **DDS_LENGTH_UNLIMITED** (p. 636); incremental_count = -1

[range] See allowed ranges in struct **DDS_AllocationSettings_t** (p. 796)

Categories

Immutable (p. 333), **Middleware Memory Management** (p. 297)

9.53.2.12 content_filtered_topic_allocation

`struct` **`DDS_AllocationSettings_t`** `DDS_DomainParticipantResourceLimitsQosPolicy::content_filtered_↩ topic_allocation`

<<***extension***>> *(p. 213)* Allocation settings applied to content filtered topic.

[default] initial_count = 4; max_count = **DDS_LENGTH_UNLIMITED** (p. 636); incremental_count = -1

[range] See allowed ranges in struct **DDS_AllocationSettings_t** (p. 796)

Categories

Immutable (p. 333), **Middleware Memory Management** (p. 297), **Content Filtering** (p. 260)

9.53.2.13 content_filter_allocation

struct **DDS_AllocationSettings_t** DDS_DomainParticipantResourceLimitsQosPolicy::content_filter_allocation

<<***extension***>> *(p. 213)* Allocation settings applied to content filter.

[default] initial_count = 4; max_count = **DDS_LENGTH_UNLIMITED** (p. 636); incremental_count = -1

[range] See allowed ranges in struct **DDS_AllocationSettings_t** (p. 796)

Categories

Immutable (p. 333), **Middleware Memory Management** (p. 297), **Content Filtering** (p. 260)

9.53.2.14 read_condition_allocation

struct **DDS_AllocationSettings_t** DDS_DomainParticipantResourceLimitsQosPolicy::read_condition_allocation

<<***extension***>> *(p. 213)* Allocation settings applied to read condition pool.

[default] initial_count = 4; max_count = **DDS_LENGTH_UNLIMITED** (p. 636), incremental_count = -1

[range] See allowed ranges in struct **DDS_AllocationSettings_t** (p. 796)

Categories

Immutable (p. 333), **Middleware Memory Management** (p. 297)

9.53.2.15 query_condition_allocation

struct **DDS_AllocationSettings_t** DDS_DomainParticipantResourceLimitsQosPolicy::query_condition_allocation

<<***extension***>> *(p. 213)* Allocation settings applied to query condition pool.

[default] initial_count = 4; max_count = **DDS_LENGTH_UNLIMITED** (p. 636), incremental_count = -1

[range] See allowed ranges in struct **DDS_AllocationSettings_t** (p. 796)

Categories

Immutable (p. 333), **Middleware Memory Management** (p. 297), **Content Filtering** (p. 260)

9.53.2.16 outstanding_asynchronous_sample_allocation

`struct` **`DDS_AllocationSettings_t`** `DDS_DomainParticipantResourceLimitsQosPolicy::outstanding_asynchronous_sample_allocation`

<<***extension***>> *(p. 213)* Allocation settings applied to the maximum number of samples (from all **DDSDataWriter** (p. 1589)) waiting to be asynchronously written.

[default] initial_count = 64; max_count = **DDS_LENGTH_UNLIMITED** (p. 636), incremental_count = -1

[range] See allowed ranges in struct **DDS_AllocationSettings_t** (p. 796)

Categories

Immutable (p. 333), **Middleware Memory Management** (p. 297), **Flow Control** (p. 282)

9.53.2.17 flow_controller_allocation

`struct` **`DDS_AllocationSettings_t`** `DDS_DomainParticipantResourceLimitsQosPolicy::flow_controller_allocation`

<<***extension***>> *(p. 213)* Allocation settings applied to flow controllers.

[default] initial_count = 4; max_count = **DDS_LENGTH_UNLIMITED** (p. 636), incremental_count = -1

[range] See allowed ranges in struct **DDS_AllocationSettings_t** (p. 796)

Categories

Immutable (p. 333), **Middleware Memory Management** (p. 297), **Flow Control** (p. 282)

9.53.2.18 local_writer_hash_buckets

`DDS_Long` `DDS_DomainParticipantResourceLimitsQosPolicy::local_writer_hash_buckets`

[DEPRECATED] (p. 384) <<***extension***>> *(p. 213)* Hash_Buckets settings applied to local DataWriters.

[default] 4

[range] [1, 10000]

Categories

Immutable (p. 333), **Middleware Memory Management** (p. 297)

9.53.2.19 local_reader_hash_buckets

```
DDS_Long DDS_DomainParticipantResourceLimitsQosPolicy::local_reader_hash_buckets
```

[DEPRECATED] (p. **384)** <<***extension***>> *(*p. *213)* Number of hash buckets for local DataReaders.

[default] 4

[range] [1, 10000]

Categories

Immutable (p. 333), **Middleware Memory Management** (p. 297)

9.53.2.20 local_publisher_hash_buckets

```
DDS_Long DDS_DomainParticipantResourceLimitsQosPolicy::local_publisher_hash_buckets
```

[DEPRECATED] (p. **384)** <<***extension***>> *(*p. *213)* Number of hash buckets for local Publisher.

[default] 1

[range] [1, 10000]

Categories

Immutable (p. 333), **Middleware Memory Management** (p. 297)

9.53.2.21 local_subscriber_hash_buckets

```
DDS_Long DDS_DomainParticipantResourceLimitsQosPolicy::local_subscriber_hash_buckets
```

[DEPRECATED] (p. **384)** <<***extension***>> *(*p. *213)* Number of hash buckets for local Subscriber.

[default] 1

[range] [1, 10000]

Categories

Immutable (p. 333), **Middleware Memory Management** (p. 297)

9.53.2.22 local_topic_hash_buckets

```
DDS_Long DDS_DomainParticipantResourceLimitsQosPolicy::local_topic_hash_buckets
```

[DEPRECATED] (p. **384)** <<***extension***>> *(*p. *213)* Number of hash buckets for local Topic.

[default] 4

[range] [1, 10000]

Categories

Immutable (p. 333), **Middleware Memory Management** (p. 297)

9.53.2.23 remote_writer_hash_buckets

```
DDS_Long DDS_DomainParticipantResourceLimitsQosPolicy::remote_writer_hash_buckets
```

[DEPRECATED] (p. **384)** <<***extension***>> *(*p. *213)* Number of hash buckets for remote DataWriters.

Remote DataWriters include all DataWriters, both local and remote.

[default] 16

[range] [1, 10000]

Categories

Immutable (p. 333), **Middleware Memory Management** (p. 297)

9.53.2.24 remote_reader_hash_buckets

```
DDS_Long DDS_DomainParticipantResourceLimitsQosPolicy::remote_reader_hash_buckets
```

[DEPRECATED] (p. **384)** <<***extension***>> *(*p. *213)* Number of hash buckets for remote DataReaders.

Remote DataReaders include all DataReaders, both local and remote.

[default] 16

[range] [1, 10000]

Categories

Immutable (p. 333), **Middleware Memory Management** (p. 297)

9.53.2.25 remote_participant_hash_buckets

```
DDS_Long DDS_DomainParticipantResourceLimitsQosPolicy::remote_participant_hash_buckets
```

[DEPRECATED] (p. **384**) <<***extension***>> *(p. 213)* Number of hash buckets for remote DomainParticipants.

Remote DomainParticipants include all DomainParticipants, both local and remote.

[default] 4

[range] [1, 10000]

Categories

Immutable (p. 333), **Middleware Memory Management** (p. 297)

9.53.2.26 matching_writer_reader_pair_hash_buckets

```
DDS_Long DDS_DomainParticipantResourceLimitsQosPolicy::matching_writer_reader_pair_hash_buckets
```

[DEPRECATED] (p. **384**) <<***extension***>> *(p. 213)* Number of hash buckets for matching local writer and remote/local reader pairs.

[default] 32

[range] [1, 10000]

Categories

Immutable (p. 333), **Middleware Memory Management** (p. 297)

9.53.2.27 matching_reader_writer_pair_hash_buckets

```
DDS_Long DDS_DomainParticipantResourceLimitsQosPolicy::matching_reader_writer_pair_hash_buckets
```

[DEPRECATED] (p. **384**) <<***extension***>> *(p. 213)* Number of hash buckets for matching local reader and remote/local writer pairs.

[default] 32

[range] [1, 10000]

Categories

Immutable (p. 333), **Middleware Memory Management** (p. 297)

9.53.2.28 ignored_entity_hash_buckets

DDS_Long DDS_DomainParticipantResourceLimitsQosPolicy::ignored_entity_hash_buckets

[DEPRECATED] (p. **384)** <<***extension***>> *(*p. *213)* Number of hash buckets for ignored entities.

[default] 1

[range] [1, 10000]

Categories

Immutable (p. 333), **Middleware Memory Management** (p. 297)

9.53.2.29 content_filtered_topic_hash_buckets

DDS_Long DDS_DomainParticipantResourceLimitsQosPolicy::content_filtered_topic_hash_buckets

[DEPRECATED] (p. **384)** <<***extension***>> *(*p. *213)* Number of hash buckets for content filtered topics.

[default] 1

[range] [1, 10000]

Categories

Immutable (p. 333), **Middleware Memory Management** (p. 297), **Content Filtering** (p. 260)

9.53.2.30 content_filter_hash_buckets

DDS_Long DDS_DomainParticipantResourceLimitsQosPolicy::content_filter_hash_buckets

[DEPRECATED] (p. **384)** <<***extension***>> *(*p. *213)* Number of hash buckets for content filters.

[default] 1

[range] [1, 10000]

Categories

Immutable (p. 333), **Middleware Memory Management** (p. 297), **Content Filtering** (p. 260)

9.53.2.31 flow_controller_hash_buckets

```
DDS_Long DDS_DomainParticipantResourceLimitsQosPolicy::flow_controller_hash_buckets
```

[DEPRECATED] (p. **384**) <<***extension***>> *(p. 213)* Number of hash buckets for flow controllers.

[default] 1

[range] [1, 10000]

Categories

Immutable (p. 333), **Middleware Memory Management** (p. 297), **Flow Control** (p. 282)

9.53.2.32 max_gather_destinations

```
DDS_Long DDS_DomainParticipantResourceLimitsQosPolicy::max_gather_destinations
```

<<***extension***>> *(p. 213)* Maximum number of destinations per RTI Connext send.

When RTI Connext sends out a message, it has the capability to send to multiple destinations to be more efficient. The maximum number of destinations per RTI Connext send is specified by max_gather_destinations.

[default] 16

[range] [16, 1 million]

Categories

Immutable (p. 333), **Middleware Memory Management** (p. 297)

9.53.2.33 participant_user_data_max_length

```
DDS_Long DDS_DomainParticipantResourceLimitsQosPolicy::participant_user_data_max_length
```

<<***extension***>> *(p. 213)* Maximum length of user data in **DDS_DomainParticipantQos** (p. 971) and **DDS_↩ ParticipantBuiltinTopicData** (p. 1229)

[default] 256

[range] [0,0x7fffffff]

Categories

Immutable (p. 333), **Middleware Memory Management** (p. 297), **Application-Defined Metadata** (p. 254)

9.53.2.34 topic_data_max_length

DDS_Long DDS_DomainParticipantResourceLimitsQosPolicy::topic_data_max_length

<<***extension***>> *(p. 213)* Maximum length of topic data in **DDS_TopicQos** (p. 1397), **DDS_TopicBuiltinTopicData** (p. 1391), **DDS_PublicationBuiltinTopicData** (p. 1264) and **DDS_SubscriptionBuiltinTopicData** (p. 1370)

[default] 256

[range] [0,0x7fffffff]

Categories

Immutable (p. 333), **Middleware Memory Management** (p. 297), **Application-Defined Metadata** (p. 254)

9.53.2.35 publisher_group_data_max_length

DDS_Long DDS_DomainParticipantResourceLimitsQosPolicy::publisher_group_data_max_length

<<***extension***>> *(p. 213)* Maximum length of group data in **DDS_PublisherQos** (p. 1276) and **DDS_Publication↩ BuiltinTopicData** (p. 1264)

[default] 256

[range] [0,0x7fffffff]

Categories

Immutable (p. 333), **Middleware Memory Management** (p. 297), **Application-Defined Metadata** (p. 254)

9.53.2.36 subscriber_group_data_max_length

DDS_Long DDS_DomainParticipantResourceLimitsQosPolicy::subscriber_group_data_max_length

<<***extension***>> *(p. 213)* Maximum length of group data in **DDS_SubscriberQos** (p. 1367) and **DDS_Subscription↩ BuiltinTopicData** (p. 1370)

[default] 256

[range] [0,0x7fffffff]

Categories

Immutable (p. 333), **Middleware Memory Management** (p. 297), **Application-Defined Metadata** (p. 254)

9.53.2.37 writer_user_data_max_length

DDS_Long DDS_DomainParticipantResourceLimitsQosPolicy::writer_user_data_max_length

<<***extension***>> *(p. 213)* Maximum length of user data in **DDS_DataWriterQos** (p. 909) and **DDS_PublicationBuiltinTopicData** (p. 1264)

[default] 256

[range] [0,0x7fffffff]

Categories

Immutable (p. 333), **Middleware Memory Management** (p. 297), **Application-Defined Metadata** (p. 254)

9.53.2.38 reader_user_data_max_length

DDS_Long DDS_DomainParticipantResourceLimitsQosPolicy::reader_user_data_max_length

<<***extension***>> *(p. 213)* Maximum length of user data in **DDS_DataReaderQos** (p. 858) and **DDS_SubscriptionBuiltinTopicData** (p. 1370)

[default] 256

[range] [0,0x7fffffff]

Categories

Immutable (p. 333), **Middleware Memory Management** (p. 297), **Application-Defined Metadata** (p. 254)

9.53.2.39 max_partitions

DDS_Long DDS_DomainParticipantResourceLimitsQosPolicy::max_partitions

<<***extension***>> *(p. 213)* Maximum number of partition name strings allowable in a **DDS_PartitionQosPolicy** (p. 1240).

This setting is made on a per DomainParticipant basis; it cannot be set individually on a per Publisher/Subscriber basis. However, the limit is enforced and applies per Publisher/Subscriber.

This value cannot exceed 64.

[default] 64

[range] [0,64]

Categories

Immutable (p. 333), **Middleware Memory Management** (p. 297), **Discovery Isolation** (p. 277)

9.53.2.40 max_partition_cumulative_characters

```
DDS_Long DDS_DomainParticipantResourceLimitsQosPolicy::max_partition_cumulative_characters
```

<<***extension***>> *(p. 213)* Maximum number of combined characters allowable in all partition names in a **DDS_↩ PartitionQosPolicy** (p. 1240).

The maximum number of combined characters should account for a terminating NULL ('\0') character for each partition name string.

This setting is made on a per DomainParticipant basis; it cannot be set individually on a per Publisher/Subscriber basis. However, the limit is enforced and applies per Publisher/Subscriber.

This value cannot exceed 256.

[default] 256

[range] [0,256]

Categories

Immutable (p. 333), **Middleware Memory Management** (p. 297), **Discovery Isolation** (p. 277)

9.53.2.41 type_code_max_serialized_length

```
DDS_Long DDS_DomainParticipantResourceLimitsQosPolicy::type_code_max_serialized_length
```

[DEPRECATED] (p. 384) <<***extension***>> *(p. 213)* Maximum size of serialized string for type code.

This parameter is an alternative to **DDS_DomainParticipantResourceLimitsQosPolicy::type_object_max_↩ serialized_length** (p. 994) for limiting the size of the type code that a **DDSDomainParticipant** (p. 1619) is able to store and propagate for user data types. Type codes can be used by external applications to understand user data types without having the data type predefined in compiled form. However, since type codes contain all of the information of a data structure, including the strings that define the names of the members of a structure, complex data structures can result in large type codes. So it is common for users to set this parameter to a large value, if used (by default, it is set to 0). However, as with all parameters in this QoS policy defining maximum sizes for variable-length elements, all DomainParticipants in the same domain should use the same value for this parameter.

Note: TypeObject v2 is now the standard, and default, method of exchanging type information in RTI Connext. It is recommended to keep **DDS_DISCOVERYCONFIG_TYPE_LOOKUP_SERVICE_CHANNEL** (p. 592) in **DDS_↩ DiscoveryConfigQosPolicy::enabled_builtin_channels** (p. 946) enabled in DomainParticipants requiring type information.

[default] 0

[range] [0,0xffff]

Categories

Immutable (p. 333), **Middleware Memory Management** (p. 297), **Type System** (p. 331)

9.53.2.42 type_object_max_serialized_length

DDS_Long DDS_DomainParticipantResourceLimitsQosPolicy::type_object_max_serialized_length

<<***extension***>> *(p. 213)* The maximum length, in bytes, that the buffer to serialize a TypeObject v1 can consume.

Type codes contain complete information about a data structure, including all member names; as a result, complex data structures can produce large type codes. This parameter limits the size of the resulting TypeObject v1 that a DomainParticipant is able to propagate.

If this parameter is set to a value between 1 and 0x7fffffff, the DomainParticipant will propagate TypeObject v1 for types that don't exceed the specified size. The desired size for a given **DDS_TypeCode** (p. 1428) can be obtained using **DDS_TypeCode::get_type_object_serialized_size** (p. 1457). If **DDS_DISCOVERYCONFIG_TYPE_LOOKUP_SERVICE_CHANNEL** (p. 592) is also enabled, both TypeObject v1 and TypeObject v2 will be propagated (TypeObject v2s are not subject to this size limit).

If set to **DDS_LENGTH_AUTO** (p. 631), the DomainParticipant will not propagate TypeObject v1. If **DDS_DISCOVERYCONFIG_TYPE_LOOKUP_SERVICE_CHANNEL** (p. 592) is also enabled, only TypeObject v2 will be propagated.

If explicitly set to 0, the DomainParticipant won't propagate any TypeObject (neither TypeObject v1 nor TypeObject v2), regardless of whether **DDS_DISCOVERYCONFIG_TYPE_LOOKUP_SERVICE_CHANNEL** (p. 592) is enabled.

Note: TypeObject v2 is now the standard, and default, method of exchanging type information in RTI Connext. It is recommended to enable **DDS_DISCOVERYCONFIG_TYPE_LOOKUP_SERVICE_CHANNEL** (p. 592) in **DDS_DiscoveryConfigQosPolicy::enabled_builtin_channels** (p. 946) in DomainParticipants requiring type validation.

[default] DDS_LENGTH_AUTO (p. 631)

[range] [0,0x7fffffff], or **DDS_LENGTH_AUTO** (p. 631)

Categories

Immutable (p. 333), **Middleware Memory Management** (p. 297), **Type System** (p. 331)

9.53.2.43 serialized_type_object_dynamic_allocation_threshold

DDS_Long DDS_DomainParticipantResourceLimitsQosPolicy::serialized_type_object_dynamic_allocation_threshold

<<***extension***>> *(p. 213)* A threshold, in bytes, for dynamic memory allocation for the serialized TypeObject v1.

Above this threshold, the memory for a TypeObject v1 is allocated dynamically. Below it, the memory is obtained from a pool of fixed-size buffers. The size of the buffers is equal to this threshold.

The effective threshold value used by the middleware is determined as follows:

- If **DDS_DomainParticipantResourceLimitsQosPolicy::type_object_max_serialized_length** (p. 994) is **DDS_LENGTH_AUTO** (p. 631) or 0, the effective threshold is 0 (no pre-allocated buffers).

- If this threshold is set to **DDS_LENGTH_AUTO** (p. 631) and **DDS_DomainParticipantResourceLimitsQos↩ Policy::type_object_max_serialized_length** (p. 994) is set to a specific value, the effective threshold is min(type_object_max_serialized_length, 8192).
- If the specified threshold exceeds type_object_max_serialized_length, it is automatically capped and adjusted down to match the maximum serialized length.

If **DDS_DISCOVERYCONFIG_TYPE_LOOKUP_SERVICE_CHANNEL** (p. 592) is also enabled, this threshold does not affect memory allocation of TypeObject v2s.

[default] DDS_LENGTH_AUTO (p. 631)

[range] [0,0x7fffffff], or **DDS_LENGTH_AUTO** (p. 631)

Categories

Immutable (p. 333), **Middleware Memory Management** (p. 297), **Type System** (p. 331)

9.53.2.44 type_object_max_deserialized_length

```
DDS_Long DDS_DomainParticipantResourceLimitsQosPolicy::type_object_max_deserialized_length
```

<<***extension***>> *(p. 213)* The maximum number of bytes that a deserialized TypeObject v1 can consume.

This parameter limits the size of the TypeObject v1 that a DomainParticipant is able to store.

Setting this field to 0 disables the ability of a DomainParticipant to receive TypeObject v1.

If **DDS_DISCOVERYCONFIG_TYPE_LOOKUP_SERVICE_CHANNEL** (p. 592) is also enabled, the DomainParticipant will always send/receive TypeObject v2 (deserialized TypeObject v2s are not subject to this size limit), but will only receive TypeObject v1 if this parameter is set to a value greater than 0.

[default] DDS_LENGTH_UNLIMITED (p. 636)

[range] [0,0x7fffffff] or **DDS_LENGTH_UNLIMITED** (p. 636)

Categories

Immutable (p. 333), **Middleware Memory Management** (p. 297), **Type System** (p. 331)

9.53.2.45 deserialized_type_object_dynamic_allocation_threshold

DDS_Long DDS_DomainParticipantResourceLimitsQosPolicy::deserialized_type_object_dynamic_allocation_threshold

<<***extension***>> *(p. 213)* A threshold, in bytes, for dynamic memory allocation for the deserialized TypeObject v1.

Above this threshold, the memory for a TypeObject v1 is allocated dynamically. Below it, the memory is obtained from a pool of fixed-size buffers. The size of the buffers is equal to this threshold.

In case **DDS_DomainParticipantResourceLimitsQosPolicy::type_object_max_deserialized_length** (p. 996) is different than **DDS_LENGTH_UNLIMITED** (p. 636) and is smaller than **DDS_DomainParticipantResourceLimitsQosPolicy::deserialized_type_object_dynamic_allocation_threshold** (p. 996), **DDS_DomainParticipantResourceLimitsQosPolicy::deserialized_type_object_dynamic_allocation_threshold** (p. 996) will be adjusted to **DDS_DomainParticipantResourceLimitsQosPolicy::type_object_max_deserialized_length** (p. 996).

If **DDS_DISCOVERYCONFIG_TYPE_LOOKUP_SERVICE_CHANNEL** (p. 592) is also enabled, this threshold does not affect memory allocation of TypeObject v2s.

[default] 4096

[range] [0,0x7fffffff] <= **DDS_DomainParticipantResourceLimitsQosPolicy::type_object_max_deserialized_length** (p. 996)

Categories

Immutable (p. 333), **Middleware Memory Management** (p. 297), **Type System** (p. 331)

9.53.2.46 contentfilter_property_max_length

DDS_Long DDS_DomainParticipantResourceLimitsQosPolicy::contentfilter_property_max_length

<<***extension***>> *(p. 213)* This field is the maximum length of all data related to a Content-filtered topic.

This is the sum of the length of the ContentFilteredTopic name, the length of the related topic name, the length of the filter expression, the length of the filter parameters, and the length of the filter name. The maximum number of combined characters should account for a terminating NULL ('\0') character for each string.

[default] 256

[range] [0,0xffff]

Categories

Immutable (p. 333), **Middleware Memory Management** (p. 297), **Content Filtering** (p. 260)

9.53.2.47 channel_seq_max_length

```
DDS_Long DDS_DomainParticipantResourceLimitsQosPolicy::channel_seq_max_length
```

<<***extension***>> *(p. 213)* Maximum number of channels that can be specified in **DDS_MultiChannelQosPolicy** (p. 1214) for MultiChannel DataWriters.

[default] 32

[range] [0,0xffff]

Categories

Immutable (p. 333), **Middleware Memory Management** (p. 297), **Content Filtering** (p. 260)

9.53.2.48 channel_filter_expression_max_length

```
DDS_Long DDS_DomainParticipantResourceLimitsQosPolicy::channel_filter_expression_max_length
```

<<***extension***>> *(p. 213)* Maximum length of a channel **DDS_ChannelSettings_t::filter_expression** (p. 822) in a MultiChannel DataWriter.

The length should account for a terminating NULL ('\0') character.

[default] 256

[range] [0,0xffff]

Categories

Immutable (p. 333), **Middleware Memory Management** (p. 297), **Content Filtering** (p. 260)

9.53.2.49 participant_property_list_max_length

```
DDS_Long DDS_DomainParticipantResourceLimitsQosPolicy::participant_property_list_max_length
```

<<***extension***>> *(p. 213)* Maximum number of properties associated with the **DDSDomainParticipant** (p. 1619).

[default] 32

[range] [0,0xffff]

Categories

Immutable (p. 333), **Middleware Memory Management** (p. 297), **Application-Defined Metadata** (p. 254)

9.53.2.50 participant_property_string_max_length

DDS_Long DDS_DomainParticipantResourceLimitsQosPolicy::participant_property_string_max_length

<<***extension***>> *(p. 213)* Maximum string length of the properties associated with the **DDSDomainParticipant** (p. 1619).

The string length is defined as the cumulative length in bytes, including the null terminating characters, of all the pairs (name, value) associated with the **DDSDomainParticipant** (p. 1619) properties.

[default] 4096

[range] [0,0xffff]

Categories

Immutable (p. 333), **Middleware Memory Management** (p. 297), **Application-Defined Metadata** (p. 254)

9.53.2.51 writer_property_list_max_length

DDS_Long DDS_DomainParticipantResourceLimitsQosPolicy::writer_property_list_max_length

<<***extension***>> *(p. 213)* Maximum number of properties associated with a **DDSDataWriter** (p. 1589).

[range] [0,0xffff]

[default] 32

Categories

Immutable (p. 333), **Middleware Memory Management** (p. 297), **Application-Defined Metadata** (p. 254)

9.53.2.52 writer_property_string_max_length

DDS_Long DDS_DomainParticipantResourceLimitsQosPolicy::writer_property_string_max_length

<<***extension***>> *(p. 213)* Maximum string length of the properties associated with a **DDSDataWriter** (p. 1589).

The string length is defined as the cumulative length in bytes, including the null terminating characters, of all the pairs (name, value) associated with the **DDSDataWriter** (p. 1589) properties.

[default] 1024

[range] [0,0xffff]

Categories

Immutable (p. 333), **Middleware Memory Management** (p. 297), **Application-Defined Metadata** (p. 254)

9.53.2.53 reader_property_list_max_length

```
DDS_Long DDS_DomainParticipantResourceLimitsQosPolicy::reader_property_list_max_length
```

<<***extension***>> *(p. 213)* Maximum number of properties associated with a **DDSDataReader** (p. 1556).

[default] 32

[range] [0,0xffff]

Categories

Immutable (p. 333), **Middleware Memory Management** (p. 297), **Application-Defined Metadata** (p. 254)

9.53.2.54 reader_property_string_max_length

```
DDS_Long DDS_DomainParticipantResourceLimitsQosPolicy::reader_property_string_max_length
```

<<***extension***>> *(p. 213)* Maximum string length of the properties associated with a **DDSDataReader** (p. 1556).

The string length is defined as the cumulative length in bytes, including the null terminating characters, of all the pairs (name, value) associated with a **DDSDataReader** (p. 1556) properties.

[default] 1024

[range] [0,0xffff]

Categories

Immutable (p. 333), **Middleware Memory Management** (p. 297), **Application-Defined Metadata** (p. 254)

9.53.2.55 max_endpoint_groups

```
DDS_Long DDS_DomainParticipantResourceLimitsQosPolicy::max_endpoint_groups
```

<<***extension***>> *(p. 213)* Maximum number of **DDS_EndpointGroup_t** (p. 1142) allowable in a **DDS_Availability↩ QosPolicy** (p. 805).

[default] 32

[range] [0,65535]

Categories

Immutable (p. 333), **Middleware Memory Management** (p. 297)

9.53.2.56 max_endpoint_group_cumulative_characters

DDS_Long DDS_DomainParticipantResourceLimitsQosPolicy::max_endpoint_group_cumulative_characters

<<***extension***>> *(p. 213)* Maximum number of combined role_name characters allowed in all **DDS_EndpointGroup_t** (p. 1142) in a **DDS_AvailabilityQosPolicy** (p. 805).

The maximum number of combined characters should account for a terminating NULL character for each role_name string.

[default] 1024

[range] [0,65535]

Categories

Immutable (p. 333), **Middleware Memory Management** (p. 297)

9.53.2.57 transport_info_list_max_length

DDS_Long DDS_DomainParticipantResourceLimitsQosPolicy::transport_info_list_max_length

<<***extension***>> *(p. 213)* Maximum number of installed transports to send and receive information about in **DDS_ParticipantBuiltinTopicData::transport_info** (p. 1232).

[default] 12

[range] [0,100]

Categories

Immutable (p. 333), **Middleware Memory Management** (p. 297), **Transports** (p. 327)

9.53.2.58 ignored_entity_replacement_kind

DDS_DomainParticipantResourceLimitsIgnoredEntityReplacementKind DDS_DomainParticipantResourceLimitsQosPolicy::ignored_entity_replacement_kind

[DEPRECATED] (p. 384) <<***extension***>> *(p. 213)* Replacement policy for the ignored entities. It sets what entity can be replaced when resource limits set in **DDS_DomainParticipantResourceLimitsQosPolicy::ignored_entity_allocation** (p. 984) are reached.

When a **DDSDomainParticipant** (p. 1619)'s number of ignored entities is greater than **DDS_DomainParticipantResourceLimitsQosPolicy::ignored_entity_allocation** (p. 984), the **DDSDomainParticipant** (p. 1619) will try to make room by replacing an existing ignored participant entry. This field specifies what entity is allowed to be replaced.

If a replaceable participant entry is not available, an out-of-resources exception will be returned.

[default] DDS_NO_REPLACEMENT_IGNORED_ENTITY_REPLACEMENT (p. 595)

See also

DDS_DomainParticipantResourceLimitsIgnoredEntityReplacementKind (p. 595)

Categories

Immutable (p. 333), **Middleware Memory Management** (p. 297)

9.53.2.59 remote_topic_query_allocation

`struct` **`DDS_AllocationSettings_t`** `DDS_DomainParticipantResourceLimitsQosPolicy::remote_topic_query_allocation`

<<***extension***>> *(p. 213)* Allocation settings applied to remote TopicQueries.

Settings applied to the allocation of information about **DDSTopicQuery** (p. 1895) objects created by other participants and discovered by this participant.

When the participant receives a new topic query that would make the current count go above `max_count`, it is not processed until the current count drops (i.e. another topic query is cancelled). The topic query stays in the Built-in ServiceRequest DataReader queue until it can be processed or it is cancelled.

[default] initial_count = 1; max_count = **DDS_LENGTH_UNLIMITED** (p. 636); incremental_count = -1

[range] See allowed ranges in struct **DDS_AllocationSettings_t** (p. 796)

Categories

Immutable (p. 333), **Middleware Memory Management** (p. 297), **Topic Queries** (p. 330)

9.53.2.60 remote_topic_query_hash_buckets

`DDS_Long` `DDS_DomainParticipantResourceLimitsQosPolicy::remote_topic_query_hash_buckets`

[DEPRECATED] (p. 384) <<***extension***>> *(p. 213)* Number of hash buckets for remote TopicQueries.

[default] 1

[range] [1, 10000]

See also

DDS_DomainParticipantResourceLimitsQosPolicy::remote_topic_query_allocation (p. 1001)

Categories

Immutable (p. 333), **Middleware Memory Management** (p. 297), **Topic Queries** (p. 330)

9.53.2.61 writer_data_tag_list_max_length

```
DDS_Long DDS_DomainParticipantResourceLimitsQosPolicy::writer_data_tag_list_max_length
```

<<***extension***>> *(p. 213)* Maximum number of data tags associated with a **DDSDataWriter** (p. 1589).

[default] 0

[range] [0,0xffff]

Categories

Immutable (p. 333), **Middleware Memory Management** (p. 297)

9.53.2.62 writer_data_tag_string_max_length

```
DDS_Long DDS_DomainParticipantResourceLimitsQosPolicy::writer_data_tag_string_max_length
```

<<***extension***>> *(p. 213)* Maximum string length of the data tags associated with a **DDSDataWriter** (p. 1589).

The string length is defined as the cumulative length in bytes, including the null terminating characters, of all the pairs (name, value) associated with the **DDSDataWriter** (p. 1589) data tags.

[default] 0

[range] [0,0xffff]

Categories

Immutable (p. 333), **Middleware Memory Management** (p. 297)

9.53.2.63 reader_data_tag_list_max_length

```
DDS_Long DDS_DomainParticipantResourceLimitsQosPolicy::reader_data_tag_list_max_length
```

<<***extension***>> *(p. 213)* Maximum number of data tags associated with a **DDSDataReader** (p. 1556).

[default] 0

[range] [0,0xffff]

Categories

Immutable (p. 333), **Middleware Memory Management** (p. 297)

9.53.2.64 reader_data_tag_string_max_length

DDS_Long DDS_DomainParticipantResourceLimitsQosPolicy::reader_data_tag_string_max_length

<<***extension***>> *(*p. *213)* Maximum string length of the data tags associated with a **DDSDataReader** (p. 1556).

The string length is defined as the cumulative length in bytes, including the null terminating characters, of all the pairs (name, value) associated with the **DDSDataReader** (p. 1556) data tags.

[default] 0

[range] [0,0xffff]

Categories

Immutable (p. 333), **Middleware Memory Management** (p. 297)

9.53.2.65 shmem_ref_transfer_mode_max_segments

DDS_UnsignedLong DDS_DomainParticipantResourceLimitsQosPolicy::shmem_ref_transfer_mode_max_segments

<<***extension***>> *(*p. *213)* Maximum number of segments created by all DataWriters belonging to a **DDSDomainParticipant** (p. 1619).

[default] 500

[range] [0,0xffffffff], but in practice, this value will be limited by the system-wide maximum number of shared memory segments. On a Linux machine, this value is provided by the kernel parameter shmmni.

Categories

Immutable (p. 333), **Middleware Memory Management** (p. 297), **Sending Large Data** (p. 290)

9.54 DDS_DoubleSeq Struct Reference

Instantiates **FooSeq** (p. 1965) < **DDS_Double** (p. 515) >

9.54.1 Detailed Description

Instantiates **FooSeq** (p. 1965) < **DDS_Double** (p. 515) >

Instantiates:

<<***generic***>> *(*p. *433)* **FooSeq** (p. 1965)

See also

DDS_Double (p. 515)

FooSeq (p. 1965)

9.55 DDS_DurabilityQosPolicy Struct Reference

<<***spec***>> *(p. 209)* This QoS policy specifies whether or not RTI Connext will store and deliver previously published data samples to new **DDSDataReader** (p. 1556) entities that join the network later.

Public Attributes

- **DDS_DurabilityQosPolicyKind kind**

  <<***basic***>> *(p. 201)* <<***spec***>> *(p. 209) The kind of durability.*
- **DDS_Boolean direct_communication**

  <<***extension***>> *(p. 213) Indicates whether or not a TRANSIENT or PERSISTENT **DDSDataReader** (p. 1556) should receive samples directly from a TRANSIENT or PERSISTENT **DDSDataWriter** (p. 1589)*
- **DDS_Long writer_depth**

  <<***basic***>> *(p. 201)* <<***extension***>> *(p. 213) Indicates the number of samples per instance that a durable **DDSDataWriter** (p. 1589) will send to a late-joining **DDSDataReader** (p. 1556).*
- struct **DDS_PersistentStorageSettings storage_settings**

  <<***extension***>> *(p. 213) Configures durable writer history and durable reader state.*

9.55.1 Detailed Description

<<***spec***>> *(p. 209)* This QoS policy specifies whether or not RTI Connext will store and deliver previously published data samples to new **DDSDataReader** (p. 1556) entities that join the network later.

Entity:

DDSTopic (p. 1885), **DDSDataReader** (p. 1556), **DDSDataWriter** (p. 1589)

Status:

DDS_OFFERED_INCOMPATIBLE_QOS_STATUS (p. 540), **DDS_REQUESTED_INCOMPATIBLE_QOS_STATUS** (p. 540)

Properties:

RxO (p. 556) = YES
Changeable (p. 556) = **UNTIL ENABLE** (p. 556)

See also

DURABILITY_SERVICE (p. 600)

9.55.2 Usage

It is possible for a **DDSDataWriter** (p. 1589) to start publishing data before all (or any) **DDSDataReader** (p. 1556) entities have joined the network.

Moreover, a **DDSDataReader** (p. 1556) that joins the network after some data has been written could potentially be interested in accessing the most current values of the data, as well as potentially some history.

This policy makes it possible for a late-joining **DDSDataReader** (p. 1556) to obtain previously published samples.

By helping to ensure that DataReaders get all data that was sent by DataWriters, regardless of when it was sent, using this QoS policy can increase system tolerance to failure conditions.

Note that although related, this does not strictly control what data RTI Connext will maintain internally. That is, RTI Connext may choose to maintain some data for its own purposes (e.g., flow control) and yet not make it available to late-joining readers if the **DURABILITY** (p. 596) policy is set to **DDS_VOLATILE_DURABILITY_QOS** (p. 598).

9.55.2.1 Transient and Persistent Durability

For the purpose of implementing the DURABILITY QoS kind TRANSIENT or PERSISTENT, RTI Connext behaves *as if* for each Topic that has **DDS_DurabilityQosPolicy::kind** (p. 1007) of **DDS_TRANSIENT_DURABILITY_QOS** (p. 599) or **DDS_PERSISTENT_DURABILITY_QOS** (p. 599) there is a corresponding "built-in" **DDSDataReader** (p. 1556) and **DDSDataWriter** (p. 1589) configured with the same DURABILITY kind. In other words, it is *as if* somewhere in the system, independent of the original **DDSDataWriter** (p. 1589), there is a built-in durable **DDSDataReader** (p. 1556) subscribing to that Topic and a built-in durable DataWriter re-publishing it as needed for the new subscribers that join the system. This functionality is provided by the *RTI Persistence Service*.

The Persistence Service can configure itself based on the QoS of your application's **DDSDataWriter** (p. 1589) and **DDSDataReader** (p. 1556) entities. For each transient or persistent **DDSTopic** (p. 1885), the built-in fictitious Persistence Service **DDSDataReader** (p. 1556) and **DDSDataWriter** (p. 1589) have their QoS configured from the QoS of your application's **DDSDataWriter** (p. 1589) and **DDSDataReader** (p. 1556) entities that communicate on that **DDSTopic** (p. 1885).

For a given **DDSTopic** (p. 1885), the usual request/offered semantics apply to the matching between any **DDSData↩ Writer** (p. 1589) in the domain that writes the **DDSTopic** (p. 1885) and the built-in transient/persistent **DDSData↩ Reader** (p. 1556) for that **DDSTopic** (p. 1885); similarly for the built-in transient/persistent **DDSDataWriter** (p. 1589) for a **DDSTopic** (p. 1885) and any **DDSDataReader** (p. 1556) for the **DDSTopic** (p. 1885). As a consequence, a **DDSData↩ Writer** (p. 1589) that has an incompatible QoS will not send its data to the *RTI Persistence Service*, and a **DDSData↩ Reader** (p. 1556) that has an incompatible QoS will not get data from it.

Incompatibilities between local **DDSDataReader** (p. 1556) and **DDSDataWriter** (p. 1589) entities and the corresponding fictitious built-in transient/persistent entities cause the **DDS_REQUESTED_INCOMPATIBLE_QOS_STATUS** (p. 540) and **DDS_OFFERED_INCOMPATIBLE_QOS_STATUS** (p. 540) to change and the corresponding Listener invocations and/or signaling of **DDSCondition** (p. 1544) objects as they would with your application's own entities.

The value of **DDS_DurabilityServiceQosPolicy::service_cleanup_delay** (p. 1010) controls when *RTI Persistence Service* is able to remove all information regarding a data instances.

Information on a data instance is maintained until the following conditions are met:

1. The instance has been explicitly disposed (instance_state = NOT_ALIVE_DISPOSED),

and

1. All samples for the disposed instance have been acknowledged, including the dispose sample itself,

and

1. A time interval longer that **DDS_DurabilityServiceQosPolicy::service_cleanup_delay** (p. 1010) has elapsed since the moment RTI Connext detected that the previous two conditions were met. (Note: Only values of zero or **DDS_DURATION_INFINITE** (p. 522) are currently supported for the service_cleanup_delay)

The utility of **DDS_DurabilityServiceQosPolicy::service_cleanup_delay** (p. 1010) is apparent in the situation where an application disposes an instance and it crashes before it has a chance to complete additional tasks related to the disposition. Upon restart, the application may ask for initial data to regain its state and the delay introduced by the service_cleanup_delay will allow the restarted application to receive the information on the disposed instance and complete the interrupted tasks.

9.55.3 Compatibility

The value offered is considered compatible with the value requested if and only if the inequality *offered kind >= requested kind* evaluates to 'TRUE'. For the purposes of this inequality, the values of DURABILITY kind are considered ordered such that **DDS_VOLATILE_DURABILITY_QOS** (p. 598) < **DDS_TRANSIENT_LOCAL_DURABILITY_QOS** (p. 599) < **DDS_TRANSIENT_DURABILITY_QOS** (p. 599) < **DDS_PERSISTENT_DURABILITY_QOS** (p. 599).

9.55.4 Member Data Documentation

9.55.4.1 kind

DDS_DurabilityQosPolicyKind DDS_DurabilityQosPolicy::kind

<<***basic***>> *(p. 201)* <<***spec***>> *(p. 209)* The kind of durability.

[default] DDS_VOLATILE_DURABILITY_QOS (p. 598)

Categories

Immutable (p. 333), **Matching** (p. 382), **Discovery Metadata** (p. 377), **Durability** (p. 278), **Data Availability** (p. 261)

9.55.4.2 direct_communication

DDS_Boolean DDS_DurabilityQosPolicy::direct_communication

<<***extension***>> *(p. 213)* Indicates whether or not a TRANSIENT or PERSISTENT **DDSDataReader** (p. 1556) should receive samples directly from a TRANSIENT or PERSISTENT **DDSDataWriter** (p. 1589)

When direct_communication is set to **DDS_BOOLEAN_TRUE** (p. 513), a TRANSIENT or PERSISTENT **DDSData↩ Reader** (p. 1556) will receive samples from both the original **DDSDataWriter** (p. 1589) configured with TRANSIENT or PERSISTENT durability and the **DDSDataWriter** (p. 1589) created by the persistence service. This peer-to-peer communication pattern provides low latency between end-points.

If the same sample is received from the original **DDSDataWriter** (p. 1589) and the persistence service, the middleware will discard the duplicate.

When direct_communication is set to **DDS_BOOLEAN_FALSE** (p. 513), a TRANSIENT or PERSISTENT **DDSData↩ Reader** (p. 1556) will only receive samples from the **DDSDataWriter** (p. 1589) created by the persistence service. This brokered communication pattern provides a way to guarantee eventual consistency.

[default] DDS_BOOLEAN_TRUE (p. 513)

Categories

Immutable (p. 333), **Discovery Metadata** (p. 377), **Durability** (p. 278), **Data Availability** (p. 261)

9.55.4.3 writer_depth

DDS_Long DDS_DurabilityQosPolicy::writer_depth

<<***basic***>> *(p. 201)* <<***extension***>> *(p. 213)* Indicates the number of samples per instance that a durable **DDSDataWriter** (p. 1589) will send to a late-joining **DDSDataReader** (p. 1556).

The default value, **DDS_AUTO_WRITER_DEPTH** (p. 599), makes this parameter equal to the following:

- **DDS_HistoryQosPolicy::depth** (p. 1164) if the history kind is **DDS_KEEP_LAST_HISTORY_QOS** (p. 604).
- **DDS_ResourceLimitsQosPolicy::max_samples_per_instance** (p. 1311) in the **DDS_ResourceLimitsQos↩ Policy** (p. 1308) if the history kind is **DDS_KEEP_ALL_HISTORY_QOS** (p. 604).

The `writer_depth` must be <= **DDS_HistoryQosPolicy::depth** (p. 1164).

`writer_depth` applies only to non-volatile DataWriters (those for which the kind is TRANSIENT_LOCAL, TRANSIENT, or PERSISTENT).

When **DDS_BatchQosPolicy::enable** (p. 810) is set to true, `writer_depth` acts as a minimum number of samples per instance that will be sent to late joiners, as opposed to the maximum. As long as a batch contains a single sample that falls within the `writer_depth` for the instance to which it belongs, then the entire batch will be sent. This means that batches sent to late-joining DataReaders may contain more samples per instance than is specified by the `writer_depth` QoS setting.

When a DataWriter responds to a TopicQuery, the samples that are evaluated against the TopicQuery filter are only those samples that fall within the `writer_depth`, not the **DDS_HistoryQosPolicy::depth** (p. 1164).

Setting `writer_depth` on the DataReader side will be ignored.

[default] DDS_AUTO_WRITER_DEPTH (p. 599)

Categories

Immutable (p. 333), **Durability** (p. 278), **Applied Per-Instance** (p. 375), **Data Availability** (p. 261)

9.55.4.4 storage_settings

```
struct DDS_PersistentStorageSettings DDS_DurabilityQosPolicy::storage_settings
```

<<***extension***>> *(p. 213)* Configures durable writer history and durable reader state.

By default, durable writer history and durable reader state are disabled. This means that a DataWriter will not persist its historical cache and a DataReader will not persist its state.

To enable durable writer history and durable reader state set **DDS_PersistentStorageSettings::enable** (p. 1244) to **DDS_BOOLEAN_TRUE** (p. 513).

Categories

Immutable (p. 333), **Durability** (p. 278), **Data Availability** (p. 261)

9.56 DDS_DurabilityServiceQosPolicy Struct Reference

<<***spec***>> *(p. 209)* Various settings to configure the external *RTI Persistence Service* used by RTI Connext for DataWriters with a **DDS_DurabilityQosPolicy** (p. 1005) setting of **DDS_PERSISTENT_DURABILITY_QOS** (p. 599) or **DDS_TRANSIENT_DURABILITY_QOS** (p. 599).

Public Attributes

- struct **DDS_Duration_t service_cleanup_delay**

  <<***spec***>> *(p. 209) Controls when the service is able to remove all information regarding a data instances.*
- **DDS_HistoryQosPolicyKind history_kind**

  <<***spec***>> *(p. 209) The kind of history to apply in recouping durable data.*
- **DDS_Long history_depth**

  <<***spec***>> *(p. 209) Setting to use for the **DDS_DurabilityQosPolicy::writer_depth** (p. 1008) when recouping durable data.*
- **DDS_Long max_samples**

  <<***spec***>> *(p. 209) Part of resource limits QoS policy to apply when feeding a late joiner.*
- **DDS_Long max_instances**

  <<***spec***>> *(p. 209) Part of resource limits QoS policy to apply when feeding a late joiner.*
- **DDS_Long max_samples_per_instance**

  <<***spec***>> *(p. 209) Part of resource limits QoS policy to apply when feeding a late joiner.*

9.56.1 Detailed Description

<<***spec***>> *(p. 209)* Various settings to configure the external *RTI Persistence Service* used by RTI Connext for DataWriters with a **DDS_DurabilityQosPolicy** (p. 1005) setting of **DDS_PERSISTENT_DURABILITY_QOS** (p. 599) or **DDS_TRANSIENT_DURABILITY_QOS** (p. 599).

Entity:

DDSTopic (p. 1885), **DDSDataWriter** (p. 1589)

Properties:

RxO (p. 556) = NO
Changeable (p. 556) = **UNTIL ENABLE** (p. 556)

See also

DURABILITY (p. 596)

HISTORY (p. 603)

RESOURCE_LIMITS (p. 635)

9.56.2 Usage

When a DataWriter's **DDS_DurabilityQosPolicy::kind** (p. 1007) is **DDS_PERSISTENT_DURABILITY_QOS** (p. 599) or **DDS_TRANSIENT_DURABILITY_QOS** (p. 599), an external service, the *RTI Persistence Service*, is used to store and possibly forward the data sent by the **DDSDataWriter** (p. 1589) to **DDSDataReader** (p. 1556) objects that are created *after* the data was initially sent.

This QoS policy is used to configure certain parameters of the Persistence Service when it operates on the behalf of the **DDSDataWriter** (p. 1589), such as how much data to store. For example, it configures the **DURABILITY** (p. 596), **HISTORY** (p. 603), and the **RESOURCE_LIMITS** (p. 635) used by the fictitious DataReader and DataWriter used by the Persistence Service. Note, however, that the Persistence Service itself may be configured to ignore these values and instead use values from its own configuration file.

9.56.3 Member Data Documentation

9.56.3.1 service_cleanup_delay

`struct` **`DDS_Duration_t`** `DDS_DurabilityServiceQosPolicy::service_cleanup_delay`

<<***spec***>> *(p. 209)* Controls when the service is able to remove all information regarding a data instances.

When the service cleanup delay is set to 0, disposed instances will be completely removed from the service. Only values of 0 and **DDS_DURATION_INFINITE** (p. 522) are currently supported.

[default] 0

Categories

Immutable (p. 333), **Discovery Metadata** (p. 377), **Durability** (p. 278), **Instance Lifecycle** (p. 286), **Data Availability** (p. 261)

9.56.3.2 history_kind

```
DDS_HistoryQosPolicyKind DDS_DurabilityServiceQosPolicy::history_kind
```

<<***spec***>> *(p. 209)* The kind of history to apply in recouping durable data.

[default] DDS_KEEP_LAST_HISTORY_QOS (p. 604)

Categories

Immutable (p. 333), **Discovery Metadata** (p. 377), **Durability** (p. 278), **Applied Per-Instance** (p. 375), **Sample Memory Management** (p. 321)

9.56.3.3 history_depth

```
DDS_Long DDS_DurabilityServiceQosPolicy::history_depth
```

<<***spec***>> *(p. 209)* Setting to use for the **DDS_DurabilityQosPolicy::writer_depth** (p. 1008) when recouping durable data.

If the **DDS_HistoryQosPolicy::depth** (p. 1164) is set to a value lower than this value, **DDS_HistoryQosPolicy::depth** (p. 1164) will be set equal to the value of this field.

[default] DDS_AUTO_WRITER_DEPTH (p. 599) (1)

Categories

Immutable (p. 333), **Discovery Metadata** (p. 377), **Durability** (p. 278), **Applied Per-Instance** (p. 375), **Sample Memory Management** (p. 321)

9.56.3.4 max_samples

```
DDS_Long DDS_DurabilityServiceQosPolicy::max_samples
```

<<***spec***>> *(p. 209)* Part of resource limits QoS policy to apply when feeding a late joiner.

[default] DDS_LENGTH_UNLIMITED (p. 636)

Categories

Immutable (p. 333), **Discovery Metadata** (p. 377), **Durability** (p. 278), **Sample Memory Management** (p. 321), **Middleware Memory Management** (p. 297), **Data Availability** (p. 261)

9.56.3.5 max_instances

```
DDS_Long DDS_DurabilityServiceQosPolicy::max_instances
```

<<***spec***>> *(*p. *209)* Part of resource limits QoS policy to apply when feeding a late joiner.

[default] DDS_LENGTH_UNLIMITED (p. 636)

Categories

Immutable (p. 333), **Discovery Metadata** (p. 377), **Durability** (p. 278), **Instance Lifecycle** (p. 286), **Sample Memory Management** (p. 321), **Middleware Memory Management** (p. 297), **Data Availability** (p. 261)

9.56.3.6 max_samples_per_instance

```
DDS_Long DDS_DurabilityServiceQosPolicy::max_samples_per_instance
```

<<***spec***>> *(*p. *209)* Part of resource limits QoS policy to apply when feeding a late joiner.

[default] DDS_LENGTH_UNLIMITED (p. 636)

Categories

Immutable (p. 333), **Discovery Metadata** (p. 377), **Durability** (p. 278), **Instance Lifecycle** (p. 286), **Sample Memory Management** (p. 321), **Middleware Memory Management** (p. 297), **Data Availability** (p. 261)

9.57 DDS_Duration_t Struct Reference

Type for *duration* representation.

Static Public Member Functions

- static **DDS_Duration_t from_micros** (**DDS_UnsignedLongLong** microseconds)

  Creates a new duration object from a duration expressed in microseconds.
- static **DDS_Duration_t from_millis** (**DDS_UnsignedLongLong** milliseconds)

  Creates a new duration object from a duration expressed in milliseconds.
- static **DDS_Duration_t from_nanos** (**DDS_UnsignedLongLong** nanoseconds)

  Creates a new duration object from a duration expressed in nanoseconds.
- static **DDS_Duration_t from_seconds** (**DDS_UnsignedLong** seconds)

  Creates a new duration object from a duration expressed in seconds.

Public Attributes

- **DDS_Long sec**
  seconds
- **DDS_UnsignedLong nanosec**
  nanoseconds

9.57.1 Detailed Description

Type for *duration* representation.

Represents a time interval.

Examples

HelloWorld_publisher.cxx, and **HelloWorld_subscriber.cxx**.

9.57.2 Member Data Documentation

9.57.2.1 sec

```
DDS_Long DDS_Duration_t::sec
```

seconds

9.57.2.2 nanosec

```
DDS_UnsignedLong DDS_Duration_t::nanosec
```

nanoseconds

[range] [0,1000000000)

9.58 DDS_DynamicData Struct Reference

A sample of any complex data type, which can be inspected and manipulated reflectively.

Public Member Functions

- **DDS_DynamicData** (const **DDS_TypeCode** *type, const **DDS_DynamicDataProperty_t** &property)

  *The constructor for new **DDS_DynamicData** (p. 1013) objects.*
- **DDS_Boolean is_valid** () const

  Indicates whether the object was constructed properly.
- **DDS_ReturnCode_t copy** (const **DDS_DynamicData** &src)

  Deeply copy from the given object to this object.
- **DDS_Boolean equal** (const **DDS_DynamicData** &other) const

  *Indicate whether the contents of another **DDS_DynamicData** (p. 1013) sample are the same as those of this one.*
- **DDS_DynamicData** & **operator=** (const **DDS_DynamicData** &src)

  Deeply copy from the given object to this object.
- **DDS_Boolean operator==** (const **DDS_DynamicData** &other) const

  *Indicate whether the contents of another **DDS_DynamicData** (p. 1013) sample are the same as those of this one.*
- **DDS_ReturnCode_t clear_all_members** ()

  Clear the contents of all data members of this object.
- **DDS_ReturnCode_t clear_optional_member** (const char *member_name, **DDS_DynamicDataMemberId** member_id)

  Clear the contents of a single optional data member of this object.
- **DDS_ReturnCode_t clear_member** (const char *member_name, **DDS_DynamicDataMemberId** member_id)

  Clear the contents of a single data member of this object.
- **DDS_Boolean is_cdr** () const

  Determine if a DynamicData object is using CDR as its data representation.
- const char * **get_cdr_buffer** (**DDS_UnsignedLong** &length) const

  Get the CDR buffer associated with a DynamicData object.
- **DDS_ReturnCode_t set_cdr_buffer** (const char *buffer, **DDS_UnsignedLong** length)

  Associate a CDR buffer with a DynamicData object.
- **DDS_ReturnCode_t from_cdr_buffer** (const char *buffer, unsigned int length)

  Deserializes a DynamicData object from a buffer of octets.
- **DDS_ReturnCode_t to_cdr_buffer** (char *buffer, unsigned int &length)

  Serializes a DynamicData object into a CDR buffer of octets.
- **DDS_ReturnCode_t to_cdr_buffer_ex** (char *buffer, unsigned int &length, **DDS_DataRepresentationId_t** representation)

  Serializes a DynamicData object into a buffer of octets.
- **DDS_ReturnCode_t to_string** (char *str, **DDS_UnsignedLong** &str_size, const **DDS_PrintFormatProperty** &property)

  Get a string representation of a DynamicData object.
- **DDS_ReturnCode_t from_string** (const char *str, **DDS_PrintFormatKind** format_kind)

  Populates a DynamicData object from a JSON string representation.
- **DDS_ReturnCode_t print** (FILE *fp, int indent) const

  Output a textual representation of this object and its contents to the given file.
- void **get_info** (**DDS_DynamicDataInfo** &info_out) const

  *Fill in the given descriptor with information about this **DDS_DynamicData** (p. 1013).*
- **DDS_ReturnCode_t bind_type** (const **DDS_TypeCode** *type)

  *If this **DDS_DynamicData** (p. 1013) object is not yet associated with a data type, set that type now to the given **DDS_TypeCode** (p. 1428).*
- **DDS_ReturnCode_t unbind_type** ()

  *Dissociate this **DDS_DynamicData** (p. 1013) object from any particular data type.*

- **DDS_ReturnCode_t bind_complex_member** (**DDS_DynamicData** &value_out, const char *member_name, **DDS_DynamicDataMemberId** member_id)

  *Use another **DDS_DynamicData** (p. 1013) object to provide access to a complex field of this **DDS_DynamicData** (p. 1013) object.*
- **DDS_ReturnCode_t unbind_complex_member** (**DDS_DynamicData** &value)

  *Tear down the association created by a **DDS_DynamicData::bind_complex_member** (p. 1042) operation, committing any changes to the outer object since then.*
- const **DDS_TypeCode** * **get_type** () const

  *Get the data type, of which this **DDS_DynamicData** (p. 1013) represents an instance.*
- **DDS_TCKind get_type_kind** () const

  Get the kind of this object's data type.
- **DDS_UnsignedLong get_member_count** () const

  Get the number of members in the type.
- **DDS_Boolean member_exists** (const char *member_name, **DDS_DynamicDataMemberId** member_id) const

  Indicates whether a member exists in this sample.
- **DDS_Boolean member_exists_in_type** (const char *member_name, **DDS_DynamicDataMemberId** member_id) const

  Indicates whether a member of a particular name/ID exists in this data sample's type.
- **DDS_ReturnCode_t get_member_info** (**DDS_DynamicDataMemberInfo** &info, const char *member_name, **DDS_DynamicDataMemberId** member_id) const

  *Fill in the given descriptor with information about the identified member of this **DDS_DynamicData** (p. 1013) sample.*
- **DDS_ReturnCode_t get_member_info_by_index** (struct **DDS_DynamicDataMemberInfo** &info, **DDS_UnsignedLong** index) const

  *Fill in the given descriptor with information about the identified member of this **DDS_DynamicData** (p. 1013) sample.*
- **DDS_ReturnCode_t get_member_type** (const **DDS_TypeCode** *&type_out, const char *member_name, **DDS_DynamicDataMemberId** member_id) const

  Get the type of the given member of this sample.
- **DDS_ReturnCode_t is_member_key** (**DDS_Boolean** &is_key_out, const char *member_name, **DDS_DynamicDataMemberId** member_id) const

  Indicates whether a given member forms part of the key of this sample's data type.
- **DDS_ReturnCode_t get_long** (**DDS_Long** &value_out, const char *member_name, **DDS_DynamicDataMemberId** member_id) const

  *Get the value of the given field, which is of type **DDS_Long** (p. 515) or another type implicitly convertible to it (**DDS_Octet** (p. 514), **DDS_Char** (p. 513), **DDS_Short** (p. 514), **DDS_UnsignedShort** (p. 514), or **DDS_Enum** (p. 516)).*
- **DDS_ReturnCode_t get_short** (**DDS_Short** &value_out, const char *member_name, **DDS_DynamicDataMemberId** member_id) const

  *Get the value of the given field, which is of type **DDS_Short** (p. 514) or another type implicitly convertible to it (**DDS_Octet** (p. 514) or **DDS_Char** (p. 513)).*
- **DDS_ReturnCode_t get_ulong** (**DDS_UnsignedLong** &value_out, const char *member_name, **DDS_DynamicDataMemberId** member_id) const

  *Get the value of the given field, which is of type **DDS_UnsignedLong** (p. 515) or another type implicitly convertible to it (**DDS_Octet** (p. 514), **DDS_Char** (p. 513), **DDS_Short** (p. 514), **DDS_UnsignedShort** (p. 514), or **DDS_Enum** (p. 516)).*
- **DDS_ReturnCode_t get_ushort** (**DDS_UnsignedShort** &value_out, const char *member_name, **DDS_DynamicDataMemberId** member_id) const

  *Get the value of the given field, which is of type **DDS_UnsignedShort** (p. 514) or another type implicitly convertible to it (**DDS_Octet** (p. 514) or **DDS_Char** (p. 513)).*
- **DDS_ReturnCode_t get_float** (**DDS_Float** &value_out, const char *member_name, **DDS_DynamicDataMemberId** member_id) const

  *Get the value of the given field, which is of type **DDS_Float** (p. 515).*

- **DDS_ReturnCode_t get_double** (**DDS_Double** &value_out, const char *member_name, **DDS_DynamicDataMemberId** member_id) const

  *Get the value of the given field, which is of type **DDS_Double** (p. 515) or another type implicitly convertible to it (**DDS_Float** (p. 515)).*

- **DDS_ReturnCode_t get_boolean** (**DDS_Boolean** &value_out, const char *member_name, **DDS_DynamicDataMemberId** member_id) const

  *Get the value of the given field, which is of type **DDS_Boolean** (p. 516).*

- **DDS_ReturnCode_t get_char** (**DDS_Char** &value_out, const char *member_name, **DDS_DynamicDataMemberId** member_id) const

  *Get the value of the given field, which is of type **DDS_Char** (p. 513).*

- **DDS_ReturnCode_t get_octet** (**DDS_Octet** &value_out, const char *member_name, **DDS_DynamicDataMemberId** member_id) const

  *Get the value of the given field, which is of type **DDS_Octet** (p. 514).*

- **DDS_ReturnCode_t get_longlong** (**DDS_LongLong** &value_out, const char *member_name, **DDS_DynamicDataMemberId** member_id) const

  *Get the value of the given field, which is of type **DDS_LongLong** (p. 515) or another type implicitly convertible to it (**DDS_Octet** (p. 514), **DDS_Char** (p. 513), **DDS_Short** (p. 514), **DDS_UnsignedShort** (p. 514), **DDS_Long** (p. 515), **DDS_UnsignedLong** (p. 515), or **DDS_Enum** (p. 516)).*

- **DDS_ReturnCode_t get_ulonglong** (**DDS_UnsignedLongLong** &value_out, const char *member_name, **DDS_DynamicDataMemberId** member_id) const

  *Get the value of the given field, which is of type **DDS_UnsignedLongLong** (p. 515) or another type implicitly convertible to it (**DDS_Octet** (p. 514), **DDS_Char** (p. 513), **DDS_Short** (p. 514), **DDS_UnsignedShort** (p. 514), **DDS_Long** (p. 515), **DDS_UnsignedLong** (p. 515), or **DDS_Enum** (p. 516)).*

- **DDS_ReturnCode_t get_longdouble** (**DDS_LongDouble** &value_out, const char *member_name, **DDS_DynamicDataMemberId** member_id) const

  *Get the value of the given field, which is of type **DDS_LongDouble** (p. 516) or another type implicitly convertible to it (**DDS_Float** (p. 515) or **DDS_Double** (p. 515)).*

- **DDS_ReturnCode_t get_wchar** (**DDS_Wchar** &value_out, const char *member_name, **DDS_DynamicDataMemberId** member_id) const

  *Get the value of the given field, which is of type **DDS_Wchar** (p. 514) or another type implicitly convertible to it (**DDS_Char** (p. 513)).*

- **DDS_ReturnCode_t get_string** (char *&value, **DDS_UnsignedLong** *size, const char *member_name, **DDS_DynamicDataMemberId** member_id) const

  Get the value of the given field, which is of type char.*

- **DDS_ReturnCode_t get_wstring** (**DDS_Wchar** *&value, **DDS_UnsignedLong** *size, const char *member_name, **DDS_DynamicDataMemberId** member_id) const

  *Get the value of the given field, which is of type **DDS_Wchar** (p. 514)*.*

- **DDS_ReturnCode_t get_int8** (**DDS_Int8** &value_out, const char *member_name, **DDS_DynamicDataMemberId** member_id) const

  *Get the value of the given field, which is of type **DDS_Int8** (p. 514).*

- **DDS_ReturnCode_t get_uint8** (**DDS_UInt8** &value_out, const char *member_name, **DDS_DynamicDataMemberId** member_id) const

  *Get the value of the given field, which is of type **DDS_UInt8** (p. 514).*

- **DDS_ReturnCode_t get_complex_member** (**DDS_DynamicData** &value_out, const char *member_name, **DDS_DynamicDataMemberId** member_id) const

  Get a copy of the value of the given field, which is of some composed type.

- **DDS_ReturnCode_t get_discriminator** (**DDS_Long** &discriminator) const

  Get the current value of the union discriminator.

- **DDS_ReturnCode_t get_long_array** (**DDS_Long** *array, **DDS_UnsignedLong** *length, const char *member_name, **DDS_DynamicDataMemberId** member_id) const

*Get a copy of the given array member. The array may contain members of type **DDS_Long** (p. 515) or **DDS_Enum** (p. 516).*

- **DDS_ReturnCode_t get_short_array** (**DDS_Short** *array, **DDS_UnsignedLong** *length, const char *member_name, **DDS_DynamicDataMemberId** member_id) const

  Get a copy of the given array member.

- **DDS_ReturnCode_t get_ulong_array** (**DDS_UnsignedLong** *array, **DDS_UnsignedLong** *length, const char *member_name, **DDS_DynamicDataMemberId** member_id) const

  *Get a copy of the given array member. The array may contain members of type **DDS_UnsignedLong** (p. 515) or **DDS_Enum** (p. 516).*

- **DDS_ReturnCode_t get_ushort_array** (**DDS_UnsignedShort** *array, **DDS_UnsignedLong** *length, const char *member_name, **DDS_DynamicDataMemberId** member_id) const

  Get a copy of the given array member.

- **DDS_ReturnCode_t get_float_array** (**DDS_Float** *array, **DDS_UnsignedLong** *length, const char *member_name, **DDS_DynamicDataMemberId** member_id) const

  Get a copy of the given array member.

- **DDS_ReturnCode_t get_double_array** (**DDS_Double** *array, **DDS_UnsignedLong** *length, const char *member_name, **DDS_DynamicDataMemberId** member_id) const

  Get a copy of the given array member.

- **DDS_ReturnCode_t get_boolean_array** (**DDS_Boolean** *array, **DDS_UnsignedLong** *length, const char *member_name, **DDS_DynamicDataMemberId** member_id) const

  Get a copy of the given array member.

- **DDS_ReturnCode_t get_char_array** (**DDS_Char** *array, **DDS_UnsignedLong** *length, const char *member_name, **DDS_DynamicDataMemberId** member_id) const

  Get a copy of the given array member.

- **DDS_ReturnCode_t get_octet_array** (**DDS_Octet** *array, **DDS_UnsignedLong** *length, const char *member_name, **DDS_DynamicDataMemberId** member_id) const

  Get a copy of the given array member.

- **DDS_ReturnCode_t get_longlong_array** (**DDS_LongLong** *array, **DDS_UnsignedLong** *length, const char *member_name, **DDS_DynamicDataMemberId** member_id) const

  Get a copy of the given array member.

- **DDS_ReturnCode_t get_ulonglong_array** (**DDS_UnsignedLongLong** *array, **DDS_UnsignedLong** *length, const char *member_name, **DDS_DynamicDataMemberId** member_id) const

  Get a copy of the given array member.

- **DDS_ReturnCode_t get_longdouble_array** (**DDS_LongDouble** *array, **DDS_UnsignedLong** *length, const char *member_name, **DDS_DynamicDataMemberId** member_id) const

  Get a copy of the given array member.

- **DDS_ReturnCode_t get_wchar_array** (**DDS_Wchar** *array, **DDS_UnsignedLong** *length, const char *member_name, **DDS_DynamicDataMemberId** member_id) const

  Get a copy of the given array member.

- **DDS_ReturnCode_t get_int8_array** (**DDS_Int8** *array, **DDS_UnsignedLong** *length, const char *member_name, **DDS_DynamicDataMemberId** member_id) const

  Get a copy of the given array member.

- **DDS_ReturnCode_t get_uint8_array** (**DDS_UInt8** *array, **DDS_UnsignedLong** *length, const char *member_name, **DDS_DynamicDataMemberId** member_id) const

  Get a copy of the given array member.

- **DDS_ReturnCode_t get_long_seq** (**DDS_LongSeq** &seq, const char *member_name, **DDS_DynamicDataMemberId** member_id) const

  Get a copy of the given sequence member.

- **DDS_ReturnCode_t get_short_seq** (**DDS_ShortSeq** &seq, const char *member_name, **DDS_DynamicDataMemberId** member_id) const

Get a copy of the given sequence member.

- **DDS_ReturnCode_t get_ulong_seq** (**DDS_UnsignedLongSeq** &seq, const char ∗member_name, **DDS_↩ DynamicDataMemberId** member_id) const

  Get a copy of the given sequence member.
- **DDS_ReturnCode_t get_ushort_seq** (**DDS_UnsignedShortSeq** &seq, const char ∗member_name, **DDS↩ _DynamicDataMemberId** member_id) const

  Get a copy of the given sequence member.
- **DDS_ReturnCode_t get_float_seq** (**DDS_FloatSeq** &seq, const char ∗member_name, **DDS_Dynamic↩ DataMemberId** member_id) const

  Get a copy of the given sequence member.
- **DDS_ReturnCode_t get_double_seq** (**DDS_DoubleSeq** &seq, const char ∗member_name, **DDS_↩ DynamicDataMemberId** member_id) const

  Get a copy of the given sequence member.
- **DDS_ReturnCode_t get_boolean_seq** (**DDS_BooleanSeq** &seq, const char ∗member_name, **DDS_↩ DynamicDataMemberId** member_id) const

  Get a copy of the given sequence member.
- **DDS_ReturnCode_t get_char_seq** (**DDS_CharSeq** &seq, const char ∗member_name, **DDS_Dynamic↩ DataMemberId** member_id) const

  Get a copy of the given sequence member.
- **DDS_ReturnCode_t get_octet_seq** (**DDS_OctetSeq** &seq, const char ∗member_name, **DDS_Dynamic↩ DataMemberId** member_id) const

  Get a copy of the given sequence member.
- **DDS_ReturnCode_t get_longlong_seq** (**DDS_LongLongSeq** &seq, const char ∗member_name, **DDS_↩ DynamicDataMemberId** member_id) const

  Get a copy of the given sequence member.
- **DDS_ReturnCode_t get_ulonglong_seq** (**DDS_UnsignedLongLongSeq** &seq, const char ∗member_name, **DDS_DynamicDataMemberId** member_id) const

  Get a copy of the given sequence member.
- **DDS_ReturnCode_t get_longdouble_seq** (**DDS_LongDoubleSeq** &seq, const char ∗member_name, **DDS↩ _DynamicDataMemberId** member_id) const

  Get a copy of the given sequence member.
- **DDS_ReturnCode_t get_wchar_seq** (**DDS_WcharSeq** &seq, const char ∗member_name, **DDS_Dynamic↩ DataMemberId** member_id) const

  Get a copy of the given sequence member.
- **DDS_ReturnCode_t get_int8_seq** (**DDS_Int8Seq** &seq, const char ∗member_name, **DDS_DynamicData↩ MemberId** member_id) const

  Get a copy of the given sequence member.
- **DDS_ReturnCode_t get_uint8_seq** (**DDS_UInt8Seq** &seq, const char ∗member_name, **DDS_Dynamic↩ DataMemberId** member_id) const

  Get a copy of the given sequence member.
- **DDS_ReturnCode_t set_long** (const char ∗member_name, **DDS_DynamicDataMemberId** member_id, **DDS_Long** value)

  *Set the value of the given field, which is of type **DDS_Long*** (p. *515*).
- **DDS_ReturnCode_t set_short** (const char ∗member_name, **DDS_DynamicDataMemberId** member_id, **DDS_Short** value)

  *Set the value of the given field, which is of type **DDS_Short*** (p. *514*).
- **DDS_ReturnCode_t set_ulong** (const char ∗member_name, **DDS_DynamicDataMemberId** member_id, **DDS_UnsignedLong** value)

  *Set the value of the given field, which is of type **DDS_UnsignedLong*** (p. *515*).

- **DDS_ReturnCode_t set_ushort** (const char *member_name, **DDS_DynamicDataMemberId** member_id, **DDS_UnsignedShort** value)

  *Set the value of the given field, which is of type **DDS_UnsignedShort** (p. 514).*
- **DDS_ReturnCode_t set_float** (const char *member_name, **DDS_DynamicDataMemberId** member_id, **DDS_Float** value)

  *Set the value of the given field, which is of type **DDS_Float** (p. 515).*
- **DDS_ReturnCode_t set_double** (const char *member_name, **DDS_DynamicDataMemberId** member_id, **DDS_Double** value)

  *Set the value of the given field, which is of type **DDS_Double** (p. 515).*
- **DDS_ReturnCode_t set_boolean** (const char *member_name, **DDS_DynamicDataMemberId** member_id, **DDS_Boolean** value)

  *Set the value of the given field, which is of type **DDS_Boolean** (p. 516).*
- **DDS_ReturnCode_t set_char** (const char *member_name, **DDS_DynamicDataMemberId** member_id, **DDS_Char** value)

  *Set the value of the given field, which is of type **DDS_Char** (p. 513).*
- **DDS_ReturnCode_t set_octet** (const char *member_name, **DDS_DynamicDataMemberId** member_id, **DDS_Octet** value)

  *Set the value of the given field, which is of type **DDS_Octet** (p. 514).*
- **DDS_ReturnCode_t set_longlong** (const char *member_name, **DDS_DynamicDataMemberId** member_id, **DDS_LongLong** value)

  *Set the value of the given field, which is of type **DDS_LongLong** (p. 515).*
- **DDS_ReturnCode_t set_ulonglong** (const char *member_name, **DDS_DynamicDataMemberId** member_id, **DDS_UnsignedLongLong** value)

  *Set the value of the given field, which is of type **DDS_UnsignedLongLong** (p. 515).*
- **DDS_ReturnCode_t set_longdouble** (const char *member_name, **DDS_DynamicDataMemberId** member↩_id, **DDS_LongDouble** value)

  *Set the value of the given field, which is of type **DDS_LongDouble** (p. 516).*
- **DDS_ReturnCode_t set_wchar** (const char *member_name, **DDS_DynamicDataMemberId** member_id, **DDS_Wchar** value)

  *Set the value of the given field, which is of type **DDS_Wchar** (p. 514).*
- **DDS_ReturnCode_t set_string** (const char *member_name, **DDS_DynamicDataMemberId** member_id, const char *value)

  Set the value of the given field of type char.*
- **DDS_ReturnCode_t set_wstring** (const char *member_name, **DDS_DynamicDataMemberId** member_id, const **DDS_Wchar** *value)

  *Set the value of the given field of type **DDS_Wchar** (p. 514)*.*
- **DDS_ReturnCode_t set_int8** (const char *member_name, **DDS_DynamicDataMemberId** member_id, **DDS_Int8** value)

  *Set the value of the given field, which is of type **DDS_Int8** (p. 514).*
- **DDS_ReturnCode_t set_uint8** (const char *member_name, **DDS_DynamicDataMemberId** member_id, **DDS_UInt8** value)

  *Set the value of the given field, which is of type **DDS_UInt8** (p. 514).*
- **DDS_ReturnCode_t set_complex_member** (const char *member_name, **DDS_DynamicDataMember↩Id** member_id, const **DDS_DynamicData** &value)

  *Copy the state of the given **DDS_DynamicData** (p. 1013) object into a member of this object.*
- **DDS_ReturnCode_t set_long_array** (const char *member_name, **DDS_DynamicDataMemberId** member_id, **DDS_UnsignedLong** length, const **DDS_Long** *array)

  *Set the contents of the given array member. The array may contain members of type **DDS_Long** (p. 515) or **DDS_Enum** (p. 516).*

- **DDS_ReturnCode_t set_short_array** (const char *member_name, **DDS_DynamicDataMemberId** member_id, **DDS_UnsignedLong** length, const **DDS_Short** *array)

  Set the contents of the given array member.
- **DDS_ReturnCode_t set_ulong_array** (const char *member_name, **DDS_DynamicDataMemberId** member_id, **DDS_UnsignedLong** length, const **DDS_UnsignedLong** *array)

  *Set the contents of the given array member. The array may contain members of type **DDS_UnsignedLong** (p. 515) or **DDS_Enum** (p. 516).*
- **DDS_ReturnCode_t set_ushort_array** (const char *member_name, **DDS_DynamicDataMemberId** member_id, **DDS_UnsignedLong** length, const **DDS_UnsignedShort** *array)

  Set the contents of the given array member.
- **DDS_ReturnCode_t set_float_array** (const char *member_name, **DDS_DynamicDataMemberId** member_id, **DDS_UnsignedLong** length, const **DDS_Float** *array)

  Set the contents of the given array member.
- **DDS_ReturnCode_t set_double_array** (const char *member_name, **DDS_DynamicDataMemberId** member_id, **DDS_UnsignedLong** length, const **DDS_Double** *array)

  Set the contents of the given array member.
- **DDS_ReturnCode_t set_boolean_array** (const char *member_name, **DDS_DynamicDataMemberId** member_id, **DDS_UnsignedLong** length, const **DDS_Boolean** *array)

  Set the contents of the given array member.
- **DDS_ReturnCode_t set_char_array** (const char *member_name, **DDS_DynamicDataMemberId** member_id, **DDS_UnsignedLong** length, const **DDS_Char** *array)

  Set the contents of the given array member.
- **DDS_ReturnCode_t set_octet_array** (const char *member_name, **DDS_DynamicDataMemberId** member_id, **DDS_UnsignedLong** length, const **DDS_Octet** *array)

  Set the contents of the given array member.
- **DDS_ReturnCode_t set_longlong_array** (const char *member_name, **DDS_DynamicDataMemberId** member_id, **DDS_UnsignedLong** length, const **DDS_LongLong** *array)

  Set the contents of the given array member.
- **DDS_ReturnCode_t set_ulonglong_array** (const char *member_name, **DDS_DynamicDataMemberId** member_id, **DDS_UnsignedLong** length, const **DDS_UnsignedLongLong** *array)

  Set the contents of the given array member.
- **DDS_ReturnCode_t set_longdouble_array** (const char *member_name, **DDS_DynamicDataMemberId** member_id, **DDS_UnsignedLong** length, const **DDS_LongDouble** *array)

  Set the contents of the given array member.
- **DDS_ReturnCode_t set_wchar_array** (const char *member_name, **DDS_DynamicDataMemberId** member_id, **DDS_UnsignedLong** length, const **DDS_Wchar** *array)

  Set the contents of the given array member.
- **DDS_ReturnCode_t set_int8_array** (const char *member_name, **DDS_DynamicDataMemberId** member_id, **DDS_UnsignedLong** length, const **DDS_Int8** *array)

  Set the contents of the given array member.
- **DDS_ReturnCode_t set_uint8_array** (const char *member_name, **DDS_DynamicDataMemberId** member_id, **DDS_UnsignedLong** length, const **DDS_UInt8** *array)

  Set the contents of the given array member.
- **DDS_ReturnCode_t set_long_seq** (const char *member_name, **DDS_DynamicDataMemberId** member_id, const **DDS_LongSeq** &value)

  Set the contents of the given sequence member.
- **DDS_ReturnCode_t set_short_seq** (const char *member_name, **DDS_DynamicDataMemberId** member_id, const **DDS_ShortSeq** &value)

  Set the contents of the given sequence member.

- **DDS_ReturnCode_t set_ulong_seq** (const char *member_name, **DDS_DynamicDataMemberId** member_id, const **DDS_UnsignedLongSeq** &value)

  Set the contents of the given sequence member.
- **DDS_ReturnCode_t set_ushort_seq** (const char *member_name, **DDS_DynamicDataMemberId** member_id, const **DDS_UnsignedShortSeq** &value)

  Set the contents of the given sequence member.
- **DDS_ReturnCode_t set_float_seq** (const char *member_name, **DDS_DynamicDataMemberId** member_id, const **DDS_FloatSeq** &value)

  Set the contents of the given sequence member.
- **DDS_ReturnCode_t set_double_seq** (const char *member_name, **DDS_DynamicDataMemberId** member_id, const **DDS_DoubleSeq** &value)

  Set the contents of the given sequence member.
- **DDS_ReturnCode_t set_boolean_seq** (const char *member_name, **DDS_DynamicDataMemberId** member_id, const **DDS_BooleanSeq** &value)

  Set the contents of the given sequence member.
- **DDS_ReturnCode_t set_char_seq** (const char *member_name, **DDS_DynamicDataMemberId** member_id, const **DDS_CharSeq** &value)

  Set the contents of the given sequence member.
- **DDS_ReturnCode_t set_octet_seq** (const char *member_name, **DDS_DynamicDataMemberId** member_id, const **DDS_OctetSeq** &value)

  Set the contents of the given sequence member.
- **DDS_ReturnCode_t set_longlong_seq** (const char *member_name, **DDS_DynamicDataMemberId** member_id, const **DDS_LongLongSeq** &value)

  Set the contents of the given sequence member.
- **DDS_ReturnCode_t set_ulonglong_seq** (const char *member_name, **DDS_DynamicDataMemberId** member_id, const **DDS_UnsignedLongLongSeq** &value)

  Set the contents of the given sequence member.
- **DDS_ReturnCode_t set_longdouble_seq** (const char *member_name, **DDS_DynamicDataMemberId** member_id, const **DDS_LongDoubleSeq** &value)

  Set the contents of the given sequence member.
- **DDS_ReturnCode_t set_wchar_seq** (const char *member_name, **DDS_DynamicDataMemberId** member_id, const **DDS_WcharSeq** &value)

  Set the contents of the given sequence member.
- **DDS_ReturnCode_t set_int8_seq** (const char *member_name, **DDS_DynamicDataMemberId** member_id, const **DDS_Int8Seq** &value)

  Set the contents of the given sequence member.
- **DDS_ReturnCode_t set_uint8_seq** (const char *member_name, **DDS_DynamicDataMemberId** member_id, const **DDS_UInt8Seq** &value)

  Set the contents of the given sequence member.
- **DDS_ReturnCode_t set_discriminator** (**DDS_Long** discriminator)

  Set the value of the union discriminator (UnionType only)
- ~**DDS_DynamicData** ()

  *Finalize and deallocate this **DDS_DynamicData*** (p. *1013*) *sample.*

9.58.1 Detailed Description

A sample of any complex data type, which can be inspected and manipulated reflectively.

Objects of type **DDS_DynamicData** (p. 1013) represent corresponding objects of the type identified by their **DDS_TypeCode** (p. 1428). Because the definition of these types may not have existed at compile time on the system on which the application is running, you will interact with the data using an API of reflective getters and setters.

For example, if you had access to your data types at compile time, you could do this:
```
theValue = theObject.theField;
```

Instead, you will do something like this:
```
theValue = get(theObject, "theField");
```

DDS_DynamicData (p. 1013) objects can represent any complex data type, including those of type kinds **DDS_TK_ARRAY** (p. 89), **DDS_TK_SEQUENCE** (p. 89), **DDS_TK_STRUCT** (p. 89), **DDS_TK_UNION** (p. 89), and **DDS_TK_VALUE** (p. 89). They cannot represent objects of basic types (e.g., integers and strings). Since those type definitions always exist on every system, you can examine their objects directly.

9.58.2 Member Names and IDs

The members of a data type can be identified in one of two ways: by their name or by their numeric ID. The former is often more transparent to human users; the latter is typically faster.

You define the name and ID of a type member when you add that member to that type. If you define your type in IDL or XML, the name will be the field name that appears in the type definition; the ID will be the one-based index of the field in declaration order. For example, in the following IDL structure, the ID of `theLong` is 2.
```
struct MyNestedType {
    char theChar;
    octet theOctetArray[10];
    long long theMultidimensionalArray[4][6][12];
    sequence<long> myArrayOfSeq[8];
};

struct MyType {
    short theShort;
    long theLong;
    MyNestedType theNestedType;
};
```

9.58.2.1 Getting and Setting Union Discriminator Values

For unions (**DDS_TK_UNION** (p. 89)), the ID of a member is the discriminator value corresponding to that member. To access the current discriminator of a union, use the **DDS_DynamicData::get_discriminator** (p. 1067) operation. Once you know the value of the discriminator, you can use it in the proper get/set_xxx() operations to access and set the member's value. Here is an example of accessing the discriminator and then using it to get a member value:
```
DynamicDataMemberId discriminatorValue = myDynamicData.get_discriminator();
DDS_Long myMemberValue = myDynamicData.get_long(NULL, discriminatorValue);
```

To get information, like the **DDS_TCKind** (p. 89), about the member that the discriminator value selects, use the **DDS_DynamicData::get_member_info** (p. 1047) API, passing in NULL for the name and the discriminator for the member ID. If the discriminator value does not select any member in the union, **DDS_DynamicData::get_member_info** (p. 1047) will return **DDS_RETCODE_NO_DATA** (p. 533).

The discriminator value is automatically set as a side-effect of setting a member using one of the set_xxx() operations. For example, given the following IDL Union:
```
union Foo switch (short) {
```

```
case 1:
    long x;
case 2:
case 3:
    string y;
};
```

Setting member "x" using the **DDS_DynamicData::set_long** (p. 1093) API, passing either "x" as the name or 1 as the member ID, will set the discriminator value to 1. When members can be selected by multiple case values, like "y" in our example, if the memeber is set by name, the first case label is chosen as the discriminator. For example, if member "y" is set by name, the discriminator value will be set to 2. To set the discriminator to a different value you can either set the member by ID, passing in the desired discriminator value, or use the **DDS_DynamicData::set_discriminator** (p. 1132) API.

Note: In past releases, accessing the discriminator value required passing in index 0 to the **DDS_DynamicData::get↩_member_info_by_index** (p. 1049) operation and then looking at the populated info.member_id. That functionality has been deprecated and replaced with the **DDS_DynamicData::get_discriminator** (p. 1067) operation.

9.58.2.2 Hierarchical Member Names

It is possible to refer to a nested member in a type without first having to use the **DDS_DynamicData::bind_complex↩_member** (p. 1042) API. You can do this by using a hierarchical name. A hierarchical member name is a concatenation of member names separated by the '.' character. The hierarchical name describes the complete path from a top-level type to the nested member. For example, in the above type, any DynamicData API that receives a member name will accept "theNestedType.theChar" to refer to the char member in MyNestedType:

```
char myChar = myDynamicData.get_char("theNestedType.theChar", DDS_DYNAMIC_DATA_MEMBER_ID_UNSPECIFIED);
```

In order to access the value of `theChar` without using a hierarchical name, you would have to first bind to theNested↩Type and then get the value:

```
myDynamicData.bind_complex_member(myBoundData, "theNestedType", DDS_DYNAMIC_DATA_MEMBER_ID_UNSPECIFIED);

DDS_Char myChar = myBoundData.get_char("theChar", DDS_DYNAMIC_DATA_MEMBER_ID_UNSPECIFIED);
```

As you can see, using a hierarchical member name removes the need to call the **DDS_DynamicData::bind_complex↩_member** (p. 1042) and **DDS_DynamicData::unbind_complex_member** (p. 1044) APIs, and allows for access to nested members at any depth directly from the top-level type.

The member name can also contain indexes to address members in arrays and sequences. For example, to set the third member in the array `theOctetArray`, you can pass in "theNestedType.theOctetArray[2]" as the member name to the **DDS_DynamicData::set_octet** (p. 1099) API. The index values when used as part of the member name are 0-based.

For multi-dimensional arrays, the indexes for each dimension should be listed comma-separated in between brackets. For example, to address a member of `theMultidimensionalArray`, the member name should be something like "theNestedType.theMultidimensionalArray[3,2,5]".

In complex types with arrays and sequences that contain other arrays and sequences, the hierarchical name may include multiple index values, one right after another. For example, in MyNestedType, `myArrayOfSeq` is an array of sequences. In order to set the third member of the sequence in the fourth member of the array, the member name would be "myNestedType.myArrayOfSeq[3][2]".

9.58.3 Arrays and Sequences

The "members" of array and sequence types, unlike those of structure and union types, don't have names or explicit member IDs. However, they may nevertheless be accessed by "ID": the ID is one more than the index. (The first element has ID 1, the second 2, etc.)

Multi-dimensional arrays are effectively flattened by this pattern. For example, for an array `theArray`[4][5], accessing ID 7 is equivalent to index 6, or the second element of the second group of 5.

To determine the length of a collection-typed member of a structure or union, you have two choices:

1. Get the length along with the data: call the appropriate array accessor (see **Getters and Setters** (p. 1025)) and check the resulting length.
2. Get the length without getting the data itself: call **DDS_DynamicData::get_member_info** (p. 1047) and check the resulting **DDS_DynamicDataMemberInfo::element_count** (p. 1136).

9.58.4 Available Functionality

The Dynamic Data API is large when measured by the number of methods it contains. But each method falls into one of a very small number of categories. You will find it easier to navigate this documentation if you understand these categories.

9.58.4.1 Lifecycle and Utility Methods

Managing the lifecycle of **DDS_DynamicData** (p. 1013) objects is simple. You have two choices:

1. Usually, you will go through a **DDSDynamicDataTypeSupport** (p. 1725) factory object, which will ensure that the type and property information for the new **DDS_DynamicData** (p. 1013) object corresponds to a registered type in your system.
2. In certain advanced cases, such as when you're navigating a nested structure, you will want to have a **DDS_↩ DynamicData** (p. 1013) object that is not bound up front to any particular type, or you will want to initialize the object in a custom way. In that case, you can call the constructor directly.

Table 9.5 Lifecycle

DDSDynamicDataTypeSupport (p. 1725)	DDS_DynamicData (p. 1013)
DDSDynamicDataTypeSupport::create_data (p. 1728)	**DDS_DynamicData::DDS_DynamicData(const DDS_TypeCode ∗, const DDS_DynamicData↩ Property_t &)** (p. 1029)
DDSDynamicDataTypeSupport::delete_data (p. 1729)	**DDS_DynamicData::∼DDS_DynamicData** (p. 1030)

You can also copy **DDS_DynamicData** (p. 1013) objects:

- **DDS_DynamicData::copy** (p. 1031)
- **DDS_DynamicData::operator=** (p. 1032)

You can test them for equality:

- **DDS_DynamicData::equal** (p. 1032)
- **DDS_DynamicData::operator==** (p. 1032)

And you can print their contents:

- **DDS_DynamicData::print** (p. 1040)
- **DDSDynamicDataTypeSupport::print_data** (p. 1729)

9.58.4.2 Getters and Setters

Most methods get or set the value of some field. These methods are named according to the type of the field they access.

The names of types vary across languages. The programming API for each language reflects that programming language. However, if your chosen language does not use the same names as the language that you used to define your types (e.g., IDL), or if you need to interoperate among programming languages, you will need to understand these differences. They are explained the following table. (Note: for modern C++, see the RTI Connext Modern C++ API reference.)

Table 9.6 Type Names Across Languages

Type	IDL	C, Traditional C++	Java	Ada
16-bit integer	short	DDS_Short	short	Standard.DDS.↩ Short
32-bit integer	long	DDS_Long	int	Standard.DDS.Long
64-bit integer	long long	DDS_LongLong	long	Standard.DDS.↩ Long_Long
Unsigned 16-bit integer	unsigned short	DDS_Unsigned↩ Short	short	Standard.DDS.↩ Unsigned_Short
Unsigned 32-bit integer	unsigned long	DDS_Unsigned↩ Long	int	Standard.DDS.Long
Unsigned 64-bit integer	unsigned long long	DDS_Unsigned↩ LongLong	long	Standard.DDS.↩ Unsigned_Long_↩ Long
float	float	DDS_Float	float	Standard.DDS.Float
double	double	DDS_Double	double	Standard.DDS.↩ Double
long double	long double	DDS_LongDouble	N/A (see CORE-14091 known issue)	Standard.DDS.↩ Long_Double
character	char	DDS_Char	char	Standard.DDS.Char
wide character	wchar	DDS_Wchar	char	Standard.DDS.↩ Wchar

Type	IDL	C, Traditional C++	Java	Ada
octet	octet	DDS_Octet	byte	Standard.DDS.Octet
boolean	boolean	DDS_Boolean	boolean	Standard.DDS.↩ Boolean
string	string	DDS_Char∗	String	Standard.DDS.↩ String
wstring	wstring	DDS_Wchar∗	String	Standard.DDS.↩ Wide_String

When working with a **DDS_DynamicData** (p. 1013) object representing an array or sequence, calling one of the "get" methods below for an index that is out of bounds will result in **DDS_RETCODE_NO_DATA** (p. 533). Calling "set" for an index that is past the end of a sequence will cause that sequence to automatically lengthen (filling with default contents).

When working with a **DDS_DynamicData** (p. 1013) object whose type contains optional members, calling one of the "get" methods below on an unset optional member or any member that is part of an unset complex optional member will result in **DDS_RETCODE_NO_DATA** (p. 533).

Table 9.7 Basic Types

Get	Set
DDS_DynamicData::get_long (p. 1051)	**DDS_DynamicData::set_long** (p. 1093)
DDS_DynamicData::get_ulong (p. 1053)	**DDS_DynamicData::set_ulong** (p. 1095)
DDS_DynamicData::get_short (p. 1052)	**DDS_DynamicData::set_short** (p. 1094)
DDS_DynamicData::get_ushort (p. 1054)	**DDS_DynamicData::set_ushort** (p. 1095)
DDS_DynamicData::get_longlong (p. 1059)	**DDS_DynamicData::set_longlong** (p. 1100)
DDS_DynamicData::get_ulonglong (p. 1060)	**DDS_DynamicData::set_ulonglong** (p. 1100)
DDS_DynamicData::get_float (p. 1055)	**DDS_DynamicData::set_float** (p. 1096)
DDS_DynamicData::get_double (p. 1056)	**DDS_DynamicData::set_double** (p. 1097)
DDS_DynamicData::get_longdouble (p. 1060)	**DDS_DynamicData::set_longdouble** (p. 1101)
DDS_DynamicData::get_boolean (p. 1056)	**DDS_DynamicData::set_boolean** (p. 1098)
DDS_DynamicData::get_octet (p. 1058)	**DDS_DynamicData::set_octet** (p. 1099)
DDS_DynamicData::get_char (p. 1057)	**DDS_DynamicData::set_char** (p. 1098)
DDS_DynamicData::get_wchar (p. 1061)	**DDS_DynamicData::set_wchar** (p. 1102)
DDS_DynamicData::get_string (p. 1062)	**DDS_DynamicData::set_string** (p. 1103)
DDS_DynamicData::get_wstring (p. 1063)	**DDS_DynamicData::set_wstring** (p. 1103)

Table 9.8 Structures, Arrays, and Other Complex Types

Get	Set
DDS_DynamicData::get_complex_member (p. 1066)	**DDS_DynamicData::set_complex_member** (p. 1105)

Table 9.9 Arrays of Basic Types

Get	Set
DDS_DynamicData::get_long_array (p. 1067)	**DDS_DynamicData::set_long_array** (p. 1107)
DDS_DynamicData::get_ulong_array (p. 1069)	**DDS_DynamicData::set_ulong_array** (p. 1109)
DDS_DynamicData::get_short_array (p. 1068)	**DDS_DynamicData::set_short_array** (p. 1108)
DDS_DynamicData::get_ushort_array (p. 1070)	**DDS_DynamicData::set_ushort_array** (p. 1109)
DDS_DynamicData::get_longlong_array (p. 1075)	**DDS_DynamicData::set_longlong_array** (p. 1115)
DDS_DynamicData::get_ulonglong_array (p. 1076)	**DDS_DynamicData::set_ulonglong_array** (p. 1115)
DDS_DynamicData::get_float_array (p. 1071)	**DDS_DynamicData::set_float_array** (p. 1110)
DDS_DynamicData::get_double_array (p. 1072)	**DDS_DynamicData::set_double_array** (p. 1111)
DDS_DynamicData::get_longdouble_array (p. 1077)	**DDS_DynamicData::set_longdouble_array** (p. 1116)
DDS_DynamicData::get_boolean_array (p. 1073)	**DDS_DynamicData::set_boolean** (p. 1098)
DDS_DynamicData::get_octet_array (p. 1075)	**DDS_DynamicData::set_octet_array** (p. 1114)
DDS_DynamicData::get_char_array (p. 1074)	**DDS_DynamicData::set_char_array** (p. 1113)
DDS_DynamicData::get_wchar_array (p. 1078)	**DDS_DynamicData::set_wchar_array** (p. 1117)

Table 9.10 Sequences of Basic Types

Get	Set
DDS_DynamicData::get_long_seq (p. 1081)	**DDS_DynamicData::set_long_seq** (p. 1120)
DDS_DynamicData::get_ulong_seq (p. 1082)	**DDS_DynamicData::set_ulong_seq** (p. 1121)
DDS_DynamicData::get_short_seq (p. 1082)	**DDS_DynamicData::set_short_seq** (p. 1121)
DDS_DynamicData::get_ushort_seq (p. 1083)	**DDS_DynamicData::set_ushort_seq** (p. 1122)
DDS_DynamicData::get_longlong_seq (p. 1088)	**DDS_DynamicData::set_longlong_seq** (p. 1127)
DDS_DynamicData::get_ulonglong_seq (p. 1089)	**DDS_DynamicData::set_ulonglong_seq** (p. 1128)
DDS_DynamicData::get_float_seq (p. 1084)	**DDS_DynamicData::set_float_seq** (p. 1123)
DDS_DynamicData::get_double_seq (p. 1085)	**DDS_DynamicData::set_double_seq** (p. 1124)
DDS_DynamicData::get_longdouble_seq (p. 1090)	**DDS_DynamicData::set_longdouble_seq** (p. 1129)
DDS_DynamicData::get_boolean_seq (p. 1086)	**DDS_DynamicData::set_boolean_seq** (p. 1125)
DDS_DynamicData::get_octet_seq (p. 1087)	**DDS_DynamicData::set_octet_seq** (p. 1126)
DDS_DynamicData::get_char_seq (p. 1087)	**DDS_DynamicData::set_char_seq** (p. 1125)
DDS_DynamicData::get_wchar_seq (p. 1091)	**DDS_DynamicData::set_wchar_seq** (p. 1129)

In addition to getting or setting a field, you can "clear" its value; that is, set it to a default zero value.

- **DDS_DynamicData::clear_all_members** (p. 1033)
- **DDS_DynamicData::clear_optional_member** (p. 1033)

9.58.4.3 Query and Iteration

Not all components of your application will have static knowledge of all of the fields of your type. Sometimes, you will want to query meta-data about the fields that appear in a given data sample.

- **DDS_DynamicData::get_type** (p. 1045)
- **DDS_DynamicData::get_type_kind** (p. 1045)
- **DDS_DynamicData::get_member_type** (p. 1050)
- **DDS_DynamicData::get_member_info** (p. 1047)
- **DDS_DynamicData::get_member_count** (p. 1045)
- **DDS_DynamicData::get_member_info_by_index** (p. 1049)
- **DDS_DynamicData::member_exists** (p. 1046)
- **DDS_DynamicData::member_exists_in_type** (p. 1047)
- **DDS_DynamicData::is_member_key** (p. 1051)

9.58.4.4 Type/Object Association

Sometimes, you may want to change the association between a data object and its type. This is not something you can do with a typical object, but with **DDS_DynamicData** (p. 1013) objects, it is a powerful capability. It allows you to, for example, examine nested structures without copying them by using a "bound" **DDS_DynamicData** (p. 1013) object as a view into an enclosing **DDS_DynamicData** (p. 1013) object.

- **DDS_DynamicData::bind_type** (p. 1041)
- **DDS_DynamicData::unbind_type** (p. 1042)
- **DDS_DynamicData::bind_complex_member** (p. 1042)
- **DDS_DynamicData::unbind_complex_member** (p. 1044)

9.58.4.5 Performance Considerations

By default, a DynamicData object stores its content in an implementation-specific representation that allows getting and setting its fields by name or ID.

When a sample is received, the CDR-formatted data is deserialized into this buffer.

When a sample is published, the data is serialized into CDR from the buffer and sent to the network.

However, there are scenarios where DynamicData is used for purposes that don't involve manipulating the object's content directly. Two common examples are:

- Recording/Replaying: Storing samples in persistent storage for later analysis or replay.
- Bridging: Transferring samples from one domain to another by transmitting samples received with a Dynamic↩ DataReader to another domain using a DynamicDataWriter.

RTI provides dedicated services, namely RTI Recording Service and RTI Routing Service, to address these use cases. However, in certain situations, you might be unable to utilize these services.

For scenarios where inspecting or modifying the content of DynamicData fields is unnecessary, DynamicData offers a high-performance method to configure a DataReader to avoid deserializing incoming samples into the internal representation. Additionally, the DynamicData API provides a method to link a CDR-formatted serialization buffer directly with a DynamicData object, facilitating direct transmission without converting the data from the internal representation to CDR.

To skip the deserialization of a DynamicData object on the DataReader side, you can set the field **DDS_Dynamic↩ DataTypeSerializationProperty_t::skip_deserialization** (p. 1141) to **DDS_BOOLEAN_TRUE** (p. 513).

Once a sample is received, you can access the serialized buffer directly by using the **DDS_DynamicData::get_cdr_↩ buffer** (p. 1035). This buffer can be stored in persistent storage in a Recording/Replaying use case.

To link a serialization buffer directly with a DynamicData object, you can use the **DDS_DynamicData::set_cdr_buffer** (p. 1036) method. This method allows you to set the buffer that will be used to serialize the DynamicData object when it is published.

MT Safety:

UNSAFE. In general, using a single **DDS_DynamicData** (p. 1013) object concurrently from multiple threads is *unsafe*.

Examples

HelloWorldPlugin.cxx.

9.58.5 Constructor & Destructor Documentation

9.58.5.1 DDS_DynamicData()

```
DDS_DynamicData::DDS_DynamicData (
            const  DDS_TypeCode * type,
            const  DDS_DynamicDataProperty_t & property )
```

The constructor for new **DDS_DynamicData** (p. 1013) objects.

The type parameter may be NULL. In that case, this **DDS_DynamicData** (p. 1013) must be *bound* with **DDS_Dynamic↩ Data::bind_type** (p. 1041) or **DDS_DynamicData::bind_complex_member** (p. 1042) before it can be used.

If the **DDS_TypeCode** (p. 1428) is not NULL, the newly constructed **DDS_DynamicData** (p. 1013) object will retain a reference to it. It is *not* safe to delete the **DDS_TypeCode** (p. 1428) until all samples that use it have themselves been deleted.

In most cases, it is not necessary to call this constructor explicitly. Instead, use **DDSDynamicDataTypeSupport↩ ::create_data** (p. 1728), and the **DDS_TypeCode** (p. 1428) and properties will be specified for you. Using the factory method also ensures that the memory management contract documented above is followed correctly, because the **DDSDynamicDataTypeSupport** (p. 1725) object maintains the **DDS_TypeCode** (p. 1428) used by the samples it creates.

However you create a **DDS_DynamicData** (p. 1013) object, you must delete it when you are finished with it. If you choose to use this constructor, delete the object with the destructor: **DDS_DynamicData::~DDS_DynamicData** (p. 1030).

```
DDS_DynamicData* sample = new DDS_DynamicData(

    myType, myProperties);

// Failure indicated by is_valid() method.

// Do something...

delete sample;
```

NOTE that RTI Connext does not explicitly generate any exceptions in this constructor, because C++ exception support is not consistent across all platforms on which RTI Connext runs. Therefore, to check whether construction succeeded, you must use the **DDS_DynamicData::is_valid** (p. 1030) method. Alternatively, you can create an **DDS_DynamicData** (p. 1013) object with **DDSDynamicDataTypeSupport::create_data** (p. 1728), which returns NULL on failure, eliminating the need to call **DDS_DynamicData::is_valid** (p. 1030).

Parameters

type	<<***in***>> *(p. 433)* The type of which the new object will represent an object.
property	<<***in***>> *(p. 433)* Properties that configure the behavior of the new object. Most users can simply use **DDS_DYNAMIC_DATA_PROPERTY_DEFAULT** (p. 105).

See also

DDS_DynamicData::is_valid (p. 1030)

DDS_DynamicData::~DDS_DynamicData (p. 1030)

DDSDynamicDataTypeSupport::create_data (p. 1728)

9.58.5.2 ~DDS_DynamicData()

```
DDS_DynamicData::~DDS_DynamicData ( )
```

Finalize and deallocate this **DDS_DynamicData** (p. 1013) sample.

MT Safety:

UNSAFE.

See also

DDS_DynamicData::DDS_DynamicData(const DDS_TypeCode *, const DDS_DynamicDataProperty_t &) (p. 1029)

9.58.6 Member Function Documentation

9.58.6.1 is_valid()

```
DDS_Boolean DDS_DynamicData::is_valid ( ) const
```

Indicates whether the object was constructed properly.

This method returns **DDS_BOOLEAN_TRUE** (p. 513) if the constructor succeeded; it returns **DDS_BOOLEAN_FALSE** (p. 513) if the constructor failed for any reason, which should also have resulted in a log message. It is only necessary to call this method if you created the **DDS_DynamicData** (p. 1013) object using the constructor, **DDS_DynamicData↩ ::DDS_DynamicData(const DDS_TypeCode ∗, const DDS_DynamicDataProperty_t &)** (p. 1029).

Possible failure reasons include passing an invalid type or invalid properties to the constructor.

This method is necessary because C++ exception support is not consistent across all of the platforms on which RTI Connext runs. Therefore, the implementation does not throw any exceptions in the constructor.

MT Safety:

UNSAFE.

See also

DDS_DynamicData::DDS_DynamicData(const DDS_TypeCode ∗, const DDS_DynamicDataProperty_t &) (p. 1029)

9.58.6.2 copy()

```
DDS_ReturnCode_t DDS_DynamicData::copy (
            const  DDS_DynamicData & src )
```

Deeply copy from the given object to this object.

MT Safety:

UNSAFE.

Exceptions

One	of the **Standard Return Codes** (p. 532) or **DDS_RETCODE_OUT_OF_RESOURCES** (p. 533)

This API is not supported when the DynamicData object is in CDR format and will fail with **DDS_RETCODE_↩ PRECONDITION_NOT_MET** (p. 532). See **DDS_DynamicData::is_cdr** (p. 1035) for more information.

See also

DDS_DynamicData::operator= (p. 1032)

9.58.6.3 equal()

```
DDS_Boolean DDS_DynamicData::equal (
            const  DDS_DynamicData & other ) const
```

Indicate whether the contents of another **DDS_DynamicData** (p. 1013) sample are the same as those of this one.

This operation compares the data and type of existing members.

This API is not supported when the DynamicData object is in CDR format and will fail with **DDS_BOOLEAN_FALSE** (p. 513). See **DDS_DynamicData::is_cdr** (p. 1035) for more information.

MT Safety:

UNSAFE.

See also

DDS_DynamicData::operator== (p. 1032)

9.58.6.4 operator=()

```
DDS_DynamicData & DDS_DynamicData::operator= (
            const  DDS_DynamicData & src )
```

Deeply copy from the given object to this object.

MT Safety:

UNSAFE.

See also

DDS_DynamicData::copy (p. 1031)

9.58.6.5 operator==()

```
DDS_Boolean DDS_DynamicData::operator== (
            const  DDS_DynamicData & other ) const
```

Indicate whether the contents of another **DDS_DynamicData** (p. 1013) sample are the same as those of this one.

This operation compares the data and type of existing members.

MT Safety:

UNSAFE.

See also

DDS_DynamicData::equal (p. 1032)

9.58.6.6 clear_all_members()

```
DDS_ReturnCode_t DDS_DynamicData::clear_all_members ( )
```

Clear the contents of all data members of this object.

MT Safety:

UNSAFE.

Exceptions

One	of the **Standard Return Codes** (p. 532)

This API is not supported when the DynamicData object is in CDR format and will fail with **DDS_RETCODE_↩ PRECONDITION_NOT_MET** (p. 532). See **DDS_DynamicData::is_cdr** (p. 1035) for more information.

See also

DDS_DynamicData::clear_optional_member (p. 1033)

9.58.6.7 clear_optional_member()

```
DDS_ReturnCode_t DDS_DynamicData::clear_optional_member (
            const char * member_name,
             DDS_DynamicDataMemberId member_id )
```

Clear the contents of a single optional data member of this object.

This method is only applicable to optional members. Members of unions, sequences, and arrays are not considered optional.

This API is not supported when the DynamicData object is in CDR format and will fail with **DDS_RETCODE_↩ PRECONDITION_NOT_MET** (p. 532). See **DDS_DynamicData::is_cdr** (p. 1035) for more information.

MT Safety:

UNSAFE.

Parameters

member_name	<<***in***>> *(p. 433)* The name of the member (to look up the member by name), or NULL (to look up the member by its ID).
member_id	<<***in***>> *(p. 433)* The ID of the member (to look up the member by its ID), or **DDS_DYNAMIC_DATA_MEMBER_ID_UNSPECIFIED** (p. 105) (to look up the member by name). See **Member Names and IDs** (p. 1022).

Exceptions

One	of the **Standard Return Codes** (p. 532)

See also

DDS_DynamicData::clear_all_members (p. 1033)

9.58.6.8 clear_member()

```
DDS_ReturnCode_t DDS_DynamicData::clear_member (
            const char * member_name,
             DDS_DynamicDataMemberId member_id )
```

Clear the contents of a single data member of this object.

This method can be used to clear both optional and non-optional members.

This API is not supported when the DynamicData object is in CDR format and will fail with **DDS_RETCODE_↩ PRECONDITION_NOT_MET** (p. 532). See **DDS_DynamicData::is_cdr** (p. 1035) for more information.

MT Safety:

UNSAFE.

Parameters

member_name	*<<**in**>> (p. 433)* The name of the member to clear or NULL to look up the member by its ID.
member_id	*<<**in**>> (p. 433)* The ID of the member to clear or **DDS_DYNAMIC_DATA_MEMBER_ID_UNSPECIFIED** (p. 105) to look up by name. See **Member Names and IDs** (p. 1022).

Exceptions

One	of the **Standard Return Codes** (p. 532)

See also

DDS_DynamicData::clear_all_members (p. 1033)

9.58.6.9 is_cdr()

```
DDS_Boolean DDS_DynamicData::is_cdr ( ) const
```

Determine if a DynamicData object is using CDR as its data representation.

This method returns **DDS_BOOLEAN_TRUE** (p. 513) if the DynamicData object is using CDR as its data representation, and **DDS_BOOLEAN_FALSE** (p. 513) otherwise.

When the DynamicData object is using CDR as its data representation, many of the DynamicData APIs cannot be used. The DynamicData object will use CDR as its data representation when **DDS_DynamicDataTypeSerialization↩ Property_t::skip_deserialization** (p. 1141) has been set to **DDS_BOOLEAN_TRUE** (p. 513) or **DDS_DynamicData↩ ::set_cdr_buffer** (p. 1036) has been used to associate a CDR buffer with the DynamicData object. See each functions's documentation for whether or not the function is supported for DynamicData objects in CDR format.

See also

DDS_DynamicData::get_cdr_buffer (p. 1035)

DDS_DynamicData::set_cdr_buffer (p. 1036)

DDS_DynamicDataTypeSerializationProperty_t::skip_deserialization (p. 1141)

9.58.6.10 get_cdr_buffer()

```
const char * DDS_DynamicData::get_cdr_buffer (
             DDS_UnsignedLong & length ) const
```

Get the CDR buffer associated with a DynamicData object.

This method returns the CDR buffer associated with a DynamicData object.

If the DynamicData object is not using CDR as its data representation, this method will return NULL and log an error.

To configure a DynamicData DataReader to not deserialize the incoming samples and keep the data in CDR format, you must set the configuration option **DDS_DynamicDataTypeSerializationProperty_t::skip_deserialization** (p. 1141) to **DDS_BOOLEAN_TRUE** (p. 513).

MT Safety:

UNSAFE.

Parameters

length	<<***out***>> *(p. 433)* CDR buffer length.

Exceptions

One	of the **Standard Return Codes** (p. 532)

9.58.6.11 set_cdr_buffer()

```
DDS_ReturnCode_t DDS_DynamicData::set_cdr_buffer (
            const char * buffer,
             DDS_UnsignedLong length )
```

Associate a CDR buffer with a DynamicData object.

This method associates a CDR buffer with a DynamicData object.

In this context, 'associates' means that the DynamicData object will use the input buffer to store its value. Additionally, if the buffer is loaned, the application is responsible for keeping it alive.

If the DynamicData is owned by the reader, this method will return an error.

MT Safety:

UNSAFE.

Parameters

buffer	*<<**in**>> (p. 433)* CDR buffer.
length	*<<**in**>> (p. 433)* CDR buffer length.

Exceptions

One	of the **Standard Return Codes** (p. 532)

9.58.6.12 from_cdr_buffer()

```
DDS_ReturnCode_t DDS_DynamicData::from_cdr_buffer (
            const char * buffer,
            unsigned int length )
```

Deserializes a DynamicData object from a buffer of octets.

This method deserializes a DynamicData object from a CDR buffer of octets.

The content of the buffer generated by the method **DDS_DynamicData::to_cdr_buffer** (p. 1037) can be provided to this method to get the DynamicData object back.

This API is not supported when the DynamicData object is in CDR format and will fail with **DDS_RETCODE_↩ PRECONDITION_NOT_MET** (p. 532). See **DDS_DynamicData::is_cdr** (p. 1035) for more information.

Parameters

buffer	*<<**in**>> (p. 433).* Deserialization buffer.
length	*<<**in**>> (p. 433).* Length of the serialized representation of the DynamicData object in the buffer.

Exceptions

One	of the **Standard Return Codes** (p. 532)

9.58.6.13 to_cdr_buffer()

```
DDS_ReturnCode_t DDS_DynamicData::to_cdr_buffer (
            char * buffer,
            unsigned int & length )
```

Serializes a DynamicData object into a CDR buffer of octets.

This method serializes a DynamicData object into a buffer of octets, using CDR as the data representation. Calling this method is equivalent to calling **DDS_DynamicData::to_cdr_buffer_ex** (p. 1038) with **DDS_AUTO_DATA_↩ REPRESENTATION** (p. 569) as the representation.

The input buffer must be large enough to store the serialized representation of the DynamicData object. Otherwise, the method will return an error code.

To determine the minimum size of the input buffer, you must call this method with the buffer set to NULL.

Parameters

buffer	*<<**out**>> (p. 433).* Serialization buffer.
length	*<<**inout**>> (p. 433).* When the buffer is set to NULL, after the method executes, length will contain the required size for the serialization buffer. When buffer is not NULL, length must contain the size of the input buffer when the method is invoked. After the method executes, length will be updated to contain the actual size of the serialized content, which may be smaller than the size obtained when `buffer` is set to NULL.

Exceptions

One	of the **Standard Return Codes** (p. 532)

9.58.6.14 to_cdr_buffer_ex()

```
DDS_ReturnCode_t DDS_DynamicData::to_cdr_buffer_ex (
            char * buffer,
            unsigned int & length,
             DDS_DataRepresentationId_t representation )
```

Serializes a DynamicData object into a buffer of octets.

This method serializes a DynamicData object into a buffer of octets using the input data representation. See **DDS_↩ DynamicData::to_cdr_buffer** (p. 1037) for details.

Parameters

buffer	*<<**out**>> (p. 433).* Serialization buffer.
length	*<<**inout**>> (p. 433).* Serialization buffer length.
representation	*<<**in**>> (p. 433).* Representation used to serialize the data

9.58.6.15 to_string()

```
DDS_ReturnCode_t DDS_DynamicData::to_string (
            char * str,
```

```
 DDS_UnsignedLong & str_size,
const  DDS_PrintFormatProperty & property )
```

Get a string representation of a DynamicData object.

This method takes a dynamic data sample and creates a string representation of the data.

The input character buffer must be big enough to store the string representation of the sample. Otherwise, the method will return an error.

To determine the minimum size of the input character buffer, the user must call this method with the buffer set to NULL.

If the size of the output string is longer than the size of an unsigned 32-bit integer, this operation will fail with **DDS_RETCODE_OUT_OF_RESOURCES** (p. 533).

This API is not supported when the DynamicData object is in CDR format and will fail with **DDS_RETCODE_PRECONDITION_NOT_MET** (p. 532). See **DDS_DynamicData::is_cdr** (p. 1035) for more information.

Parameters

str	*<<**out**>> (p. 433).* Output string representing the dynamic data sample.
str_size	*<<**inout**>> (p. 433).* When `str` is set to NULL, after the method executes, `str_size` will contain a buffer size big enough to hold the string representation of the data. When `str` is not NULL, `str_size` must contain the size of the input buffer when the method is invoked. If the size of the input buffer is too small, after the method executes, `str_size` will be updated to contain the required size of the string content and the method will return **DDS_RETCODE_OUT_OF_RESOURCES** (p. 533).
property	*<<**in**>> (p. 433).* Properties describing what the format of the output string should be. Cannot be NULL.

Exceptions

One	of the **Standard Return Codes** (p. 532), **DDS_RETCODE_OUT_OF_RESOURCES** (p. 533)

9.58.6.16 from_string()

```
DDS_ReturnCode_t DDS_DynamicData::from_string (
            const char * str,
             DDS_PrintFormatKind format_kind )
```

Populates a DynamicData object from a JSON string representation.

This method takes a JSON string representing a data sample of this type, and uses it to populate this DynamicData.

Parameters

str	*<<**in**>> (p. 433).* The JSON string representation
format_kind	*<<**in**>> (p. 433).* Must be set to **DDS_JSON_PRINT_FORMAT** (p. 67), the only currently supported format.

Exceptions

One	of the **Standard Return Codes** (p. 532), **DDS_RETCODE_OUT_OF_RESOURCES** (p. 533)

9.58.6.17 print()

```
DDS_ReturnCode_t DDS_DynamicData::print (
            FILE * fp,
            int indent ) const
```

Output a textual representation of this object and its contents to the given file.

This method is equivalent to **DDSDynamicDataTypeSupport::print_data** (p. 1729).

This API is not supported when the DynamicData object is in CDR format and will fail with **DDS_RETCODE_↩ PRECONDITION_NOT_MET** (p. 532). See **DDS_DynamicData::is_cdr** (p. 1035) for more information.

Precondition

The type must be an aggregation or collection. That is, its **DDS_TCKind** (p. 89) must be one of: **DDS_TK_↩ STRUCT** (p. 89), **DDS_TK_VALUE** (p. 89), **DDS_TK_UNION** (p. 89), **DDS_TK_SEQUENCE** (p. 89) or **DDS_TK↩ _ARRAY** (p. 89), otherwise this function fails with **DDS_RETCODE_PRECONDITION_NOT_MET** (p. 532).

MT Safety:

UNSAFE.

Parameters

fp	*<<**in**>> (p. 433)* The file into which the object should be printed (to print to standard output, provide the stream pointer 'stdout')
indent	*<<**in**>> (p. 433)* The output of this method will be pretty-printed. This argument indicates the amount of initial indentation of the output.

Exceptions

One	of the **Standard Return Codes** (p. 532)

See also

DDSDynamicDataTypeSupport::print_data (p. 1729)

9.58.6.18 get_info()

```
void DDS_DynamicData::get_info (
            DDS_DynamicDataInfo & info_out ) const
```

Fill in the given descriptor with information about this **DDS_DynamicData** (p. 1013).

This API is not supported when the DynamicData object is in CDR format and will print an error log. See **DDS_↩ DynamicData::is_cdr** (p. 1035) for more information.

MT Safety:

UNSAFE.

Parameters

info_out	<<***out***>> *(p. 433)* The descriptor object whose contents will be overwritten by this operation.

9.58.6.19 bind_type()

```
DDS_ReturnCode_t DDS_DynamicData::bind_type (
            const  DDS_TypeCode * type )
```

If this **DDS_DynamicData** (p. 1013) object is not yet associated with a data type, set that type now to the given **DDS↩ _TypeCode** (p. 1428).

This advanced operation allows you to reuse a single **DDS_DynamicData** (p. 1013) object with multiple data types.

This API is not supported when the DynamicData object is in CDR format and will fail with **DDS_RETCODE_↩ PRECONDITION_NOT_MET** (p. 532). See **DDS_DynamicData::is_cdr** (p. 1035) for more information.

```
DDS_DynamicData* myData = new DDS_DynamicData(NULL, myProperties);

DDS_TypeCode* myType = ...;

myData->bind_type(myType);

// Do something...

myData->unbind_type();

delete myData;
```

Note that the **DDS_DynamicData** (p. 1013) object will retain a reference to the **DDS_TypeCode** (p. 1428) object you provide. It is *not* safe to delete the **DDS_TypeCode** (p. 1428) until after it is unbound.

MT Safety:

UNSAFE.

Parameters

type	<<***in***>> *(p. 433)* The type to associate with this **DDS_DynamicData** (p. 1013) object.

Exceptions

One	of the **Standard Return Codes** (p. 532)

See also

DDS_DynamicData::unbind_type (p. 1042)

9.58.6.20 unbind_type()

```
DDS_ReturnCode_t DDS_DynamicData::unbind_type ( )
```

Dissociate this **DDS_DynamicData** (p. 1013) object from any particular data type.

This step is necessary before the object can be associated with a new data type.

This operation clears all members as a side effect.

This API is not supported when the DynamicData object is in CDR format and will fail with **DDS_RETCODE_↩ PRECONDITION_NOT_MET** (p. 532). See **DDS_DynamicData::is_cdr** (p. 1035) for more information.

MT Safety:

UNSAFE.

Exceptions

One	of the **Standard Return Codes** (p. 532)

See also

DDS_DynamicData::bind_type (p. 1041)

DDS_DynamicData::clear_all_members (p. 1033)

9.58.6.21 bind_complex_member()

```
DDS_ReturnCode_t DDS_DynamicData::bind_complex_member (
            DDS_DynamicData & value_out,
```

```
        const char * member_name,
         DDS_DynamicDataMemberId member_id )
```

Use another **DDS_DynamicData** (p. 1013) object to provide access to a complex field of this **DDS_DynamicData** (p. 1013) object.

This API is not supported when the DynamicData object is in CDR format and will fail with **DDS_RETCODE_↩ PRECONDITION_NOT_MET** (p. 532). See **DDS_DynamicData::is_cdr** (p. 1035) for more information.

For example, consider the following data types:

```
struct MyFieldType {

    float theFloat;

};


struct MyOuterType {

    MyFieldType complexMember;

};
```

Suppose you have an instance of `MyOuterType`, and you would like to examine the contents of its member `complexMember`. To do this, you must *bind* another **DDS_DynamicData** (p. 1013) object to that member. This operation will bind the type code of the member to the provided **DDS_DynamicData** (p. 1013) object and perform additional initialization.

The following example demonstrates the usage pattern. Note that error handling has been omitted for brevity.

```
DDS_DynamicData outer = ...;

DDS_DynamicData toBeBound(NULL, myProperties);

outer.bind_complex_member(

    toBeBound,

    "complexMember",

    DDS_DYNAMIC_DATA_MEMBER_ID_UNSPECIFIED);

float theFloatValue = 0.0f;

toBeBound.get_float(

    theFloat,

    "theFloat"

    DDS_DYNAMIC_DATA_MEMBER_ID_UNSPECIFIED);

outer.unbind_complex_member(toBeBound);
```

This operation is only permitted when the object toBeBound (named as in the example above) is not currently associated with any type, including already being bound to another member. You can see in the example that this object is created directly with the constructer and is not provided with a **DDS_TypeCode** (p. 1428).

Only a single member of a given **DDS_DynamicData** (p. 1013) object may be bound at one time – however, members *of* members may be recursively bound to any depth. Furthermore, while the outer object has a bound member, it may only be modified through that bound member. That is, after calling this member, all "set" operations on the outer object will be disabled until **DDS_DynamicData::unbind_complex_member** (p. 1044) has been called. Furthermore, any bound member must be unbound before a sample can be written or deleted.

This method is logically related to **DDS_DynamicData::get_complex_member** (p. 1066) in that both allow you to examine the state of nested objects. They are different in an important way: this method provides a view into an outer

object, such that any change made to the inner object will be reflected in the outer. But the **DDS_DynamicData::get_complex_member** (p. 1066) operation *copies* the state of the nested object; changes to it will not be reflected in the source object.

Note that you can bind to a member of a sequence at an index that is past the current length of that sequence. In that case, this method behaves like a "set" method: it automatically lengthens the sequence (filling in default elements) to allow the bind to take place. See **Getters and Setters** (p. 1025).

MT Safety:

UNSAFE.

Parameters

value_out	<<***out***>> *(p. 433)* The object that you wish to bind to the field.
member_name	<<***in***>> *(p. 433)* The name of the member or NULL to look up the member by its ID.
member_id	<<***in***>> *(p. 433)* The ID of the member or **DDS_DYNAMIC_DATA_MEMBER_ID_UNSPECIFIED** (p. 105) to look up by name. See **Member Names and IDs** (p. 1022).

Exceptions

One	of the **Standard Return Codes** (p. 532)

See also

DDS_DynamicData::unbind_complex_member (p. 1044)

DDS_DynamicData::get_complex_member (p. 1066)

9.58.6.22 unbind_complex_member()

```
DDS_ReturnCode_t DDS_DynamicData::unbind_complex_member (
            DDS_DynamicData & value )
```

Tear down the association created by a **DDS_DynamicData::bind_complex_member** (p. 1042) operation, committing any changes to the outer object since then.

Some changes to the outer object will not be observable until after you have performed this operation.

If you have called **DDS_DynamicData::bind_complex_member** (p. 1042) on a data sample, you must unbind before writing or deleting the sample.

This API is not supported when the DynamicData object is in CDR format and will fail with **DDS_RETCODE_PRECONDITION_NOT_MET** (p. 532). See **DDS_DynamicData::is_cdr** (p. 1035) for more information.

MT Safety:

UNSAFE.

Parameters

value	*<<**in**>> (p. 433)* The same object you passed to **DDS_DynamicData::bind_complex_member** (p. 1042). This argument is used for error checking purposes.

Exceptions

One	of the **Standard Return Codes** (p. 532) or **DDS_RETCODE_OUT_OF_RESOURCES** (p. 533)

See also

DDS_DynamicData::bind_complex_member (p. 1042)

9.58.6.23 get_type()

```
const  DDS_TypeCode * DDS_DynamicData::get_type ( ) const
```

Get the data type, of which this **DDS_DynamicData** (p. 1013) represents an instance.

MT Safety:

UNSAFE.

9.58.6.24 get_type_kind()

```
DDS_TCKind DDS_DynamicData::get_type_kind ( ) const
```

Get the kind of this object's data type.

This is a convenience method. It's equivalent to calling **DDS_DynamicData::get_type** (p. 1045) followed by **DDS_↩ TypeCode::kind** (p. 1432).

MT Safety:

UNSAFE.

9.58.6.25 get_member_count()

```
DDS_UnsignedLong DDS_DynamicData::get_member_count ( ) const
```

Get the number of members in the type.

For objects of type kind **DDS_TK_ARRAY** (p. 89) or **DDS_TK_SEQUENCE** (p. 89), this method returns the number of elements in the collection.

For objects of type kind **DDS_TK_STRUCT** (p. 89) or **DDS_TK_VALUE** (p. 89), it returns the number of fields in the type.

For objects of type kind **DDS_TK_UNION** (p. 89), it returns 1 if the current discriminator selects a member in the union, and 0 if it does not.

This API is not supported when the DynamicData object is in CDR format and will fail with **DDS_RETCODE_↩ PRECONDITION_NOT_MET** (p. 532). See **DDS_DynamicData::is_cdr** (p. 1035) for more information.

MT Safety:

UNSAFE.

See also

DDS_DynamicData::get_member_info_by_index (p. 1049)

9.58.6.26 member_exists()

```
DDS_Boolean DDS_DynamicData::member_exists (
            const char * member_name,
             DDS_DynamicDataMemberId member_id ) const
```

Indicates whether a member exists in this sample.

You only need to specify the name OR the ID (not both).

If the member doesn't exist in the type, this function returns false. In all other cases, it provides the same result as **DDS↩ _DynamicDataMemberInfo::member_exists** (p. 1136), which is retrieved with **DDS_DynamicData::get_member_↩ info** (p. 1047).

For union types, if a member can be selected with multiple discriminator values, looking up if a member exists using any of the values that select the member will return the same result.

This API is not supported when the DynamicData object is in CDR format and will fail with **DDS_BOOLEAN_FALSE** (p. 513). See **DDS_DynamicData::is_cdr** (p. 1035) for more information.

MT Safety:

UNSAFE.

Parameters

member_name	<<***in***>> *(p. 433)* The name of the member or NULL to look up the member by its ID.
member_id	<<***in***>> *(p. 433)* The ID of the member or **DDS_DYNAMIC_DATA_MEMBER_ID_UNSPECIFIED** (p. 105) to look up by name. See **Member Names and IDs** (p. 1022).

See also

DDS_DynamicData::member_exists_in_type (p. 1047)

DDS_DYNAMIC_DATA_MEMBER_ID_UNSPECIFIED (p. 105)

9.58.6.27 member_exists_in_type()

```
DDS_Boolean DDS_DynamicData::member_exists_in_type (
            const char * member_name,
             DDS_DynamicDataMemberId member_id ) const
```

Indicates whether a member of a particular name/ID exists in this data sample's type.

You only need to specify the name OR the ID (not both).

This API is not supported when the DynamicData object is in CDR format and will fail with **DDS_BOOLEAN_FALSE** (p. 513). See **DDS_DynamicData::is_cdr** (p. 1035) for more information.

MT Safety:

UNSAFE.

Parameters

member_name	<<***in***>> *(p. 433)* The name of the member or NULL to look up the member by its ID.
member_id	<<***in***>> *(p. 433)* The ID of the member or **DDS_DYNAMIC_DATA_MEMBER_ID_UNSPECIFIED** (p. 105) to look up by name. See **Member Names and IDs** (p. 1022).

See also

DDS_DynamicData::member_exists (p. 1046)

DDS_DYNAMIC_DATA_MEMBER_ID_UNSPECIFIED (p. 105)

9.58.6.28 get_member_info()

```
DDS_ReturnCode_t DDS_DynamicData::get_member_info (
            DDS_DynamicDataMemberInfo & info,
           const char * member_name,
            DDS_DynamicDataMemberId member_id ) const
```

Fill in the given descriptor with information about the identified member of this **DDS_DynamicData** (p. 1013) sample.

This API is not supported when the DynamicData object is in CDR format and will fail with **DDS_RETCODE_↩ PRECONDITION_NOT_MET** (p. 532). See **DDS_DynamicData::is_cdr** (p. 1035) for more information.

This operation is valid for objects of **DDS_TCKind** (p. 89) **DDS_TK_ARRAY** (p. 89), **DDS_TK_SEQUENCE** (p. 89), **DDS_TK_STRUCT** (p. 89), **DDS_TK_UNION** (p. 89), and **DDS_TK_VALUE** (p. 89).

MT Safety:

UNSAFE.

When this sample represents a struct, a value type, or a union:

- If the specified member is not defined in the type, this function fails with **DDS_RETCODE_NO_DATA** (p. 533). This includes cases when a union discriminator does not select a member in the union.
- If the specified member is defined in the type but doesn't exist in this data sample, this function returns an object with **DDS_DynamicDataMemberInfo::member_exists** (p. 1136) set to false.
- If the specified member is defined in the type and exists in this data sample, **DDS_DynamicDataMemberInfo↩ ::member_exists** (p. 1136) is true.

When this sample represents a sequence and `member_id` is the 1-based element index:

- If `member_id` is greater than the sequence's maximum length, this function fails with **DDS_RETCODE_NO_↩ DATA** (p. 533).
- If `member_id` is greater than the sequence's current length but smaller than or equal to its maximum length, this function returns an object with **DDS_DynamicDataMemberInfo::member_exists** (p. 1136) set to false.
- If `member_id` is smaller than or equal to the current length, **DDS_DynamicDataMemberInfo::member_exists** (p. 1136) is true.

When this sample represents an array, this function either fails with with **DDS_RETCODE_NO_DATA** (p. 533) when the index is out of bounds or else returns an object with **DDS_DynamicDataMemberInfo::member_exists** (p. 1136) set to true.

Parameters

info	<<***out***>> *(p. 433)* The descriptor object whose contents will be overwritten by this operations.
member_name	<<***in***>> *(p. 433)* The name of the member for which to get the info or NULL to look up the member by its ID. Only one of the name and the ID may be unspecified.
member_id	<<***in***>> *(p. 433)* The ID of the member for which to get the info, or **DDS_DYNAMIC_DATA_MEMBER_ID_UNSPECIFIED** (p. 105) to look up by name. See **Member Names and IDs** (p. 1022).

The **DDS_DynamicDataMemberInfo** (p. 1135) returned will contain information about the identified member, including its type, whether it exists in this sample, and other metadata.

The bounds for strings and collections are not provided in the returned **DDS_DynamicDataMemberInfo** (p. 1135). To access that information you must query the **DDS_TypeCode** (p. 1428) of the member's type. See **DDS_↩ TypeCode::length** (p. 1448), **DDS_TypeCode::array_dimension_count** (p. 1449), and **DDS_TypeCode::array_↩ dimension** (p. 1450).

The **DDS_TypeCode** (p. 1428) of a DynamicData object can be accessed using **DDS_DynamicData::get_type** (p. 1045). You must iterate through the TypeCode to locate the members TypeCode. Once obtained, you can query the bounds using the TypeCode APIs mentioned above.

Exceptions

One	of the **Standard Return Codes** (p. 532)

See also

DDS_DynamicData::get_member_info_by_index (p. 1049)

DDS_DYNAMIC_DATA_MEMBER_ID_UNSPECIFIED (p. 105)

9.58.6.29 get_member_info_by_index()

```
DDS_ReturnCode_t DDS_DynamicData::get_member_info_by_index (
            struct  DDS_DynamicDataMemberInfo & info,
             DDS_UnsignedLong index ) const
```

Fill in the given descriptor with information about the identified member of this **DDS_DynamicData** (p. 1013) sample.

This API is not supported when the DynamicData object is in CDR format and will fail with **DDS_RETCODE_↩ PRECONDITION_NOT_MET** (p. 532). See **DDS_DynamicData::is_cdr** (p. 1035) for more information.

This operation is valid for objects of **DDS_TCKind** (p. 89) **DDS_TK_ARRAY** (p. 89), **DDS_TK_SEQUENCE** (p. 89), **DDS_TK_STRUCT** (p. 89), **DDS_TK_VALUE** (p. 89), and **DDS_TK_UNION** (p. 89).

Using this API to retrieve the discriminator value of a union by passing in index 0 has been deprecated. You should use the **DDS_DynamicData::get_discriminator** (p. 1067) API instead. This API can still retrieve the discriminator value as long as the discriminator selects a member in the union. However, if the current discriminator value does not select a member in the union, this API will return **DDS_RETCODE_NO_DATA** (p. 533).

MT Safety:

UNSAFE.

Parameters

info	*<<**out**>> (p. 433)* The descriptor object whose contents will be overwritten by this operations.
index	*<<**in**>> (p. 433)* The zero-base of the member for which to get the info.

Exceptions

One	of the **Standard Return Codes** (p. 532)

See also

DDS_DynamicData::get_member_info (p. 1047)

DDS_DynamicData::get_member_count (p. 1045)

DDS_DynamicData::get_discriminator (p. 1067)

DDS_DYNAMIC_DATA_MEMBER_ID_UNSPECIFIED (p. 105)

9.58.6.30 get_member_type()

```
DDS_ReturnCode_t DDS_DynamicData::get_member_type (
            const  DDS_TypeCode *& type_out,
            const char * member_name,
             DDS_DynamicDataMemberId member_id ) const
```

Get the type of the given member of this sample.

The member can be looked up either by name or by ID.

This operation is valid for objects of **DDS_TCKind** (p. 89) **DDS_TK_ARRAY** (p. 89), **DDS_TK_SEQUENCE** (p. 89), **DDS_TK_STRUCT** (p. 89), and **DDS_TK_VALUE** (p. 89). For type kinds **DDS_TK_ARRAY** (p. 89) and **DDS_TK_↩ SEQUENCE** (p. 89), the index into the collection is taken to be one less than the ID, if specified. If this index is valid, this operation will return the content type of this collection.

This API is not supported when the DynamicData object is in CDR format and will fail with **DDS_RETCODE_↩ PRECONDITION_NOT_MET** (p. 532). See **DDS_DynamicData::is_cdr** (p. 1035) for more information.

MT Safety:

UNSAFE.

Parameters

type_out	<<***out***>> *(p. 433)* If this method returned success, this argument refers to the found member's type.
member_name	<<***in***>> *(p. 433)* The name of the member or NULL to look up the member by its ID.
member_id	<<***in***>> *(p. 433)* The ID of the member or **DDS_DYNAMIC_DATA_MEMBER_ID_UNSPECIFIED** (p. 105) to look up by name. See **Member Names and IDs** (p. 1022).

Exceptions

One	of the **Standard Return Codes** (p. 532)

See also

DDS_DynamicData::get_member_info (p. 1047)

DDS_DYNAMIC_DATA_MEMBER_ID_UNSPECIFIED (p. 105)

9.58.6.31 is_member_key()

```
DDS_ReturnCode_t DDS_DynamicData::is_member_key (
             DDS_Boolean & is_key_out,
            const char * member_name,
             DDS_DynamicDataMemberId member_id ) const
```

Indicates whether a given member forms part of the key of this sample's data type.

This operation is only valid for samples of types of kind **DDS_TK_STRUCT** (p. 89) or **DDS_TK_VALUE** (p. 89).

This API is not supported when the DynamicData object is in CDR format and will fail with **DDS_RETCODE_↩ PRECONDITION_NOT_MET** (p. 532). See **DDS_DynamicData::is_cdr** (p. 1035) for more information.

MT Safety:

UNSAFE.

Parameters

is_key_out	<<***out***>> *(p. 433)* If this method returned success, this argument indicates whether the indicated member is part of the key.
member_name	<<***in***>> *(p. 433)* The name of the member or NULL to look up the member by its ID.
member_id	<<***in***>> *(p. 433)* The ID of the member or **DDS_DYNAMIC_DATA_MEMBER_ID_UNSPECIFIED** (p. 105) to look up by name. See **Member Names and IDs** (p. 1022).

Exceptions

One	of the **Standard Return Codes** (p. 532)

9.58.6.32 get_long()

```
DDS_ReturnCode_t DDS_DynamicData::get_long (
            DDS_Long & value_out,
           const char * member_name,
            DDS_DynamicDataMemberId member_id ) const
```

Get the value of the given field, which is of type **DDS_Long** (p. 515) or another type implicitly convertible to it (**DDS_Octet** (p. 514), **DDS_Char** (p. 513), **DDS_Short** (p. 514), **DDS_UnsignedShort** (p. 514), or **DDS_Enum** (p. 516)).

The member may be specified by name or by ID.

This API is not supported when the DynamicData object is in CDR format and will fail with **DDS_RETCODE_PRECONDITION_NOT_MET** (p. 532). See **DDS_DynamicData::is_cdr** (p. 1035) for more information.

MT Safety:

UNSAFE.

Parameters

value_out	<<***out***>> *(p. 433)* If this method returned success, this argument will contain the value of the indicated member.
member_name	<<***in***>> *(p. 433)* The name of the member or NULL to look up the member by its ID.
member_id	<<***in***>> *(p. 433)* The ID of the member or **DDS_DYNAMIC_DATA_MEMBER_ID_UNSPECIFIED** (p. 105) to look up by name. See **Member Names and IDs** (p. 1022).

Exceptions

One	of the **Standard Return Codes** (p. 532). If the member is optional and not set, this operation will return **DDS_RETCODE_NO_DATA** (p. 533)

See also

DDS_DynamicData::set_long (p. 1093)

9.58.6.33 get_short()

```
DDS_ReturnCode_t DDS_DynamicData::get_short (
            DDS_Short & value_out,
           const char * member_name,
            DDS_DynamicDataMemberId member_id ) const
```

Get the value of the given field, which is of type **DDS_Short** (p. 514) or another type implicitly convertible to it (**DDS_Octet** (p. 514) or **DDS_Char** (p. 513)).

The member may be specified by name or by ID.

This API is not supported when the DynamicData object is in CDR format and will fail with **DDS_RETCODE_↩ PRECONDITION_NOT_MET** (p. 532). See **DDS_DynamicData::is_cdr** (p. 1035) for more information.

MT Safety:

UNSAFE.

Parameters

value_out	*<<**out**>> (p. 433)* If this method returned success, this argument will contain the value of the indicated member.
member_name	*<<**in**>> (p. 433)* The name of the member or NULL to look up the member by its ID.
member_id	*<<**in**>> (p. 433)* The ID of the member or **DDS_DYNAMIC_DATA_MEMBER_ID_UNSPECIFIED** (p. 105) to look up by name. See **Member Names and IDs** (p. 1022).

Exceptions

One	of the **Standard Return Codes** (p. 532). If the member is optional and not set, this operation will return **DDS_RETCODE_NO_DATA** (p. 533)

See also

DDS_DynamicData::set_short (p. 1094)

9.58.6.34 get_ulong()

```
DDS_ReturnCode_t DDS_DynamicData::get_ulong (
            DDS_UnsignedLong & value_out,
           const char * member_name,
            DDS_DynamicDataMemberId member_id ) const
```

Get the value of the given field, which is of type **DDS_UnsignedLong** (p. 515) or another type implicitly convertible to it (**DDS_Octet** (p. 514), **DDS_Char** (p. 513), **DDS_Short** (p. 514), **DDS_UnsignedShort** (p. 514), or **DDS_Enum** (p. 516)).

The member may be specified by name or by ID.

This API is not supported when the DynamicData object is in CDR format and will fail with **DDS_RETCODE_↩ PRECONDITION_NOT_MET** (p. 532). See **DDS_DynamicData::is_cdr** (p. 1035) for more information.

MT Safety:

UNSAFE.

Parameters

value_out	<<***out***>> *(p. 433)* If this method returned success, this argument will contain the value of the indicated member.
member_name	<<***in***>> *(p. 433)* The name of the member or NULL to look up the member by its ID.
member_id	<<***in***>> *(p. 433)* The ID of the member or **DDS_DYNAMIC_DATA_MEMBER_ID_UNSPECIFIED** (p. 105) to look up by name. See **Member Names and IDs** (p. 1022).

Exceptions

One	of the **Standard Return Codes** (p. 532). If the member is optional and not set, this operation will return **DDS_RETCODE_NO_DATA** (p. 533)

See also

DDS_DynamicData::set_ulong (p. 1095)

9.58.6.35 get_ushort()

```
DDS_ReturnCode_t DDS_DynamicData::get_ushort (
             DDS_UnsignedShort & value_out,
            const char * member_name,
             DDS_DynamicDataMemberId member_id ) const
```

Get the value of the given field, which is of type **DDS_UnsignedShort** (p. 514) or another type implicitly convertible to it (**DDS_Octet** (p. 514) or **DDS_Char** (p. 513)).

The member may be specified by name or by ID.

This API is not supported when the DynamicData object is in CDR format and will fail with **DDS_RETCODE_↩ PRECONDITION_NOT_MET** (p. 532). See **DDS_DynamicData::is_cdr** (p. 1035) for more information.

MT Safety:

UNSAFE.

Parameters

value_out	<<***out***>> *(p. 433)* If this method returned success, this argument will contain the value of the indicated member.
member_name	<<***in***>> *(p. 433)* The name of the member or NULL to look up the member by its ID.
member_id	<<***in***>> *(p. 433)* The ID of the member or **DDS_DYNAMIC_DATA_MEMBER_ID_UNSPECIFIED** (p. 105) to look up by name. See **Member Names and IDs** (p. 1022).

Exceptions

One	of the **Standard Return Codes** (p. 532). If the member is optional and not set, this operation will return **DDS_RETCODE_NO_DATA** (p. 533)

See also

DDS_DynamicData::set_ushort (p. 1095)

9.58.6.36 get_float()

```
DDS_ReturnCode_t DDS_DynamicData::get_float (
            DDS_Float & value_out,
           const char * member_name,
            DDS_DynamicDataMemberId member_id ) const
```

Get the value of the given field, which is of type **DDS_Float** (p. 515).

The member may be specified by name or by ID.

This API is not supported when the DynamicData object is in CDR format and will fail with **DDS_RETCODE_↩ PRECONDITION_NOT_MET** (p. 532). See **DDS_DynamicData::is_cdr** (p. 1035) for more information.

MT Safety:

UNSAFE.

Parameters

value_out	<<***out***>> *(p. 433)* If this method returned success, this argument will contain the value of the indicated member.
member_name	<<***in***>> *(p. 433)* The name of the member or NULL to look up the member by its ID.
member_id	<<***in***>> *(p. 433)* The ID of the member or **DDS_DYNAMIC_DATA_MEMBER_ID_UNSPECIFIED** (p. 105) to look up by name. See **Member Names and IDs** (p. 1022).

Exceptions

One	of the **Standard Return Codes** (p. 532). If the member is optional and not set, this operation will return **DDS_RETCODE_NO_DATA** (p. 533)

See also

DDS_DynamicData::set_float (p. 1096)

9.58.6.37 get_double()

```
DDS_ReturnCode_t DDS_DynamicData::get_double (
            DDS_Double & value_out,
           const char * member_name,
            DDS_DynamicDataMemberId member_id ) const
```

Get the value of the given field, which is of type **DDS_Double** (p. 515) or another type implicitly convertible to it (**DDS_Float** (p. 515)).

The member may be specified by name or by ID.

This API is not supported when the DynamicData object is in CDR format and will fail with **DDS_RETCODE_PRECONDITION_NOT_MET** (p. 532). See **DDS_DynamicData::is_cdr** (p. 1035) for more information.

MT Safety:

UNSAFE.

Parameters

value_out	*<<**out**>> (p. 433)* If this method returned success, this argument will contain the value of the indicated member.
member_name	*<<**in**>> (p. 433)* The name of the member or NULL to look up the member by its ID.
member_id	*<<**in**>> (p. 433)* The ID of the member or **DDS_DYNAMIC_DATA_MEMBER_ID_UNSPECIFIED** (p. 105) to look up by name. See **Member Names and IDs** (p. 1022).

Exceptions

One	of the **Standard Return Codes** (p. 532). If the member is optional and not set, this operation will return **DDS_RETCODE_NO_DATA** (p. 533)

See also

DDS_DynamicData::set_double (p. 1097)

9.58.6.38 get_boolean()

```
DDS_ReturnCode_t DDS_DynamicData::get_boolean (
            DDS_Boolean & value_out,
```

```
        const char * member_name,
         DDS_DynamicDataMemberId member_id ) const
```

Get the value of the given field, which is of type **DDS_Boolean** (p. 516).

The member may be specified by name or by ID.

This API is not supported when the DynamicData object is in CDR format and will fail with **DDS_RETCODE_↩ PRECONDITION_NOT_MET** (p. 532). See **DDS_DynamicData::is_cdr** (p. 1035) for more information.

MT Safety:

UNSAFE.

Parameters

value_out	<<***out***>> *(p. 433)* If this method returned success, this argument will contain the value of the indicated member.
member_name	<<***in***>> *(p. 433)* The name of the member or NULL to look up the member by its ID.
member_id	<<***in***>> *(p. 433)* The ID of the member or **DDS_DYNAMIC_DATA_MEMBER_ID_UNSPECIFIED** (p. 105) to look up by name. See **Member Names and IDs** (p. 1022).

Exceptions

One	of the **Standard Return Codes** (p. 532). If the member is optional and not set, this operation will return **DDS_RETCODE_NO_DATA** (p. 533)

See also

DDS_DynamicData::set_boolean (p. 1098)

9.58.6.39 get_char()

```
DDS_ReturnCode_t DDS_DynamicData::get_char (
             DDS_Char & value_out,
            const char * member_name,
             DDS_DynamicDataMemberId member_id ) const
```

Get the value of the given field, which is of type **DDS_Char** (p. 513).

The member may be specified by name or by ID.

This API is not supported when the DynamicData object is in CDR format and will fail with **DDS_RETCODE_↩ PRECONDITION_NOT_MET** (p. 532). See **DDS_DynamicData::is_cdr** (p. 1035) for more information.

MT Safety:

UNSAFE.

Parameters

value_out	<<***out***>> *(p. 433)* If this method returned success, this argument will contain the value of the indicated member.
member_name	<<***in***>> *(p. 433)* The name of the member or NULL to look up the member by its ID.
member_id	<<***in***>> *(p. 433)* The ID of the member or **DDS_DYNAMIC_DATA_MEMBER_ID_UNSPECIFIED** (p. 105) to look up by name. See **Member Names and IDs** (p. 1022).

Exceptions

One	of the **Standard Return Codes** (p. 532). If the member is optional and not set, this operation will return **DDS_RETCODE_NO_DATA** (p. 533)

See also

DDS_DynamicData::set_char (p. 1098)

9.58.6.40 get_octet()

```
DDS_ReturnCode_t DDS_DynamicData::get_octet (
             DDS_Octet & value_out,
            const char * member_name,
             DDS_DynamicDataMemberId member_id ) const
```

Get the value of the given field, which is of type **DDS_Octet** (p. 514).

The member may be specified by name or by ID.

This API is not supported when the DynamicData object is in CDR format and will fail with **DDS_RETCODE_↩ PRECONDITION_NOT_MET** (p. 532). See **DDS_DynamicData::is_cdr** (p. 1035) for more information.

MT Safety:

UNSAFE.

Parameters

value_out	<<***out***>> *(p. 433)* If this method returned success, this argument will contain the value of the indicated member.
member_name	<<***in***>> *(p. 433)* The name of the member or NULL to look up the member by its ID.
member_id	<<***in***>> *(p. 433)* The ID of the member or **DDS_DYNAMIC_DATA_MEMBER_ID_UNSPECIFIED** (p. 105) to look up by name. See **Member Names and IDs** (p. 1022).

Exceptions

One	of the **Standard Return Codes** (p. 532). If the member is optional and not set, this operation will return **DDS_RETCODE_NO_DATA** (p. 533)

See also

DDS_DynamicData::set_octet (p. 1099)

9.58.6.41 get_longlong()

```
DDS_ReturnCode_t DDS_DynamicData::get_longlong (
            DDS_LongLong & value_out,
           const char * member_name,
            DDS_DynamicDataMemberId member_id ) const
```

Get the value of the given field, which is of type **DDS_LongLong** (p. 515) or another type implicitly convertible to it (**DDS_Octet** (p. 514), **DDS_Char** (p. 513), **DDS_Short** (p. 514), **DDS_UnsignedShort** (p. 514), **DDS_Long** (p. 515), **DDS_UnsignedLong** (p. 515), or **DDS_Enum** (p. 516)).

The member may be specified by name or by ID.

This API is not supported when the DynamicData object is in CDR format and will fail with **DDS_RETCODE_↩ PRECONDITION_NOT_MET** (p. 532). See **DDS_DynamicData::is_cdr** (p. 1035) for more information.

MT Safety:

UNSAFE.

Parameters

value_out	<<***out***>> *(p. 433)* If this method returned success, this argument will contain the value of the indicated member.
member_name	<<***in***>> *(p. 433)* The name of the member or NULL to look up the member by its ID.
member_id	<<***in***>> *(p. 433)* The ID of the member or **DDS_DYNAMIC_DATA_MEMBER_ID_UNSPECIFIED** (p. 105) to look up by name. See **Member Names and IDs** (p. 1022).

Exceptions

One	of the **Standard Return Codes** (p. 532). If the member is optional and not set, this operation will return **DDS_RETCODE_NO_DATA** (p. 533)

See also

DDS_DynamicData::set_longlong (p. 1100)

9.58.6.42 get_ulonglong()

```
DDS_ReturnCode_t DDS_DynamicData::get_ulonglong (
            DDS_UnsignedLongLong & value_out,
           const char * member_name,
            DDS_DynamicDataMemberId member_id ) const
```

Get the value of the given field, which is of type **DDS_UnsignedLongLong** (p. 515) or another type implicitly convertible to it (**DDS_Octet** (p. 514), **DDS_Char** (p. 513), **DDS_Short** (p. 514), **DDS_UnsignedShort** (p. 514), **DDS_Long** (p. 515), **DDS_UnsignedLong** (p. 515), or **DDS_Enum** (p. 516)).

The member may be specified by name or by ID.

This API is not supported when the DynamicData object is in CDR format and will fail with **DDS_RETCODE_↩ PRECONDITION_NOT_MET** (p. 532). See **DDS_DynamicData::is_cdr** (p. 1035) for more information.

MT Safety:

UNSAFE.

Parameters

value_out	*<<**out**>> (p. 433)* If this method returned success, this argument will contain the value of the indicated member.
member_name	*<<**in**>> (p. 433)* The name of the member or NULL to look up the member by its ID.
member_id	*<<**in**>> (p. 433)* The ID of the member or **DDS_DYNAMIC_DATA_MEMBER_ID_UNSPECIFIED** (p. 105) to look up by name. See **Member Names and IDs** (p. 1022).

Exceptions

One	of the **Standard Return Codes** (p. 532). If the member is optional and not set, this operation will return **DDS_RETCODE_NO_DATA** (p. 533)

See also

DDS_DynamicData::set_ulonglong (p. 1100)

9.58.6.43 get_longdouble()

```
DDS_ReturnCode_t DDS_DynamicData::get_longdouble (
            DDS_LongDouble & value_out,
           const char * member_name,
            DDS_DynamicDataMemberId member_id ) const
```

Get the value of the given field, which is of type **DDS_LongDouble** (p. 516) or another type implicitly convertible to it (**DDS_Float** (p. 515) or **DDS_Double** (p. 515)).

The member may be specified by name or by ID.

This API is not supported when the DynamicData object is in CDR format and will fail with **DDS_RETCODE_↩ PRECONDITION_NOT_MET** (p. 532). See **DDS_DynamicData::is_cdr** (p. 1035) for more information.

MT Safety:

UNSAFE.

Parameters

value_out	<<***out***>> *(p. 433)* If this method returned success, this argument will contain the value of the indicated member.
member_name	<<***in***>> *(p. 433)* The name of the member or NULL to look up the member by its ID.
member_id	<<***in***>> *(p. 433)* The ID of the member or **DDS_DYNAMIC_DATA_MEMBER_ID_UNSPECIFIED** (p. 105) to look up by name. See **Member Names and IDs** (p. 1022).

Exceptions

One	of the **Standard Return Codes** (p. 532). If the member is optional and not set, this operation will return **DDS_RETCODE_NO_DATA** (p. 533)

See also

DDS_DynamicData::set_longdouble (p. 1101)

9.58.6.44 get_wchar()

```
DDS_ReturnCode_t DDS_DynamicData::get_wchar (
            DDS_Wchar & value_out,
           const char * member_name,
            DDS_DynamicDataMemberId member_id ) const
```

Get the value of the given field, which is of type **DDS_Wchar** (p. 514) or another type implicitly convertible to it (**DDS↩ _Char** (p. 513)).

The member may be specified by name or by ID.

This API is not supported when the DynamicData object is in CDR format and will fail with **DDS_RETCODE_↩ PRECONDITION_NOT_MET** (p. 532). See **DDS_DynamicData::is_cdr** (p. 1035) for more information.

MT Safety:

UNSAFE.

Parameters

value_out	*<<**out**>> (p. 433)* If this method returned success, this argument will contain the value of the indicated member.
member_name	*<<**in**>> (p. 433)* The name of the member or NULL to look up the member by its ID.
member_id	*<<**in**>> (p. 433)* The ID of the member or **DDS_DYNAMIC_DATA_MEMBER_ID_UNSPECIFIED** (p. 105) to look up by name. See **Member Names and IDs** (p. 1022).

Exceptions

One	of the **Standard Return Codes** (p. 532). If the member is optional and not set, this operation will return **DDS_RETCODE_NO_DATA** (p. 533)

See also

DDS_DynamicData::set_wchar (p. 1102)

9.58.6.45 get_string()

```
DDS_ReturnCode_t DDS_DynamicData::get_string (
            char *& value,
             DDS_UnsignedLong * size,
            const char * member_name,
             DDS_DynamicDataMemberId member_id ) const
```

Get the value of the given field, which is of type char∗.

The member may be specified by name or by ID.

This API is not supported when the DynamicData object is in CDR format and will fail with **DDS_RETCODE_↩ PRECONDITION_NOT_MET** (p. 532). See **DDS_DynamicData::is_cdr** (p. 1035) for more information.

MT Safety:

UNSAFE.

Parameters

value	<<***out***>> *(*p. *433)* The string into which the middleware should copy the string member. The allocated size of this string is indicated by `size` argument. If the size is sufficient to hold the contents of the member, they will be copied into this string. If `value` is a pointer to NULL, the middleware will allocate a new string for you of sufficient length; it will be your responsibility to free that string. If the size is insufficient but greater than zero, the middleware will *not* free your string; instead, this operation will fail and `size` will contain the minimum required size.
size	<<***inout***>> *(*p. *433)* As an input argument, the allocated size of the string. As an output argument, the actual size of the contents.
member_name	<<***in***>> *(*p. *433)* The name of the member or NULL to look up the member by its ID.
member_id	<<***in***>> *(*p. *433)* The ID of the member or **DDS_DYNAMIC_DATA_MEMBER_ID_UNSPECIFIED** (p. 105) to look up by name. See **Member Names and IDs** (p. 1022).

Exceptions

One	of the **Standard Return Codes** (p. 532). If the member is optional and not set, this operation will return **DDS_RETCODE_NO_DATA** (p. 533). This operation may also return **DDS_RETCODE_OUT_OF_RESOURCES** (p. 533).

See also

DDS_String_alloc (p. 759)

DDS_String_free (p. 762)

DDS_DynamicData::set_string (p. 1103)

9.58.6.46 get_wstring()

```
DDS_ReturnCode_t DDS_DynamicData::get_wstring (
            DDS_Wchar *& value,
            DDS_UnsignedLong * size,
           const char * member_name,
            DDS_DynamicDataMemberId member_id ) const
```

Get the value of the given field, which is of type **DDS_Wchar** (p. 514)*.

The member may be specified by name or by ID.

This API is not supported when the DynamicData object is in CDR format and will fail with **DDS_RETCODE_↩ PRECONDITION_NOT_MET** (p. 532). See **DDS_DynamicData::is_cdr** (p. 1035) for more information.

MT Safety:

UNSAFE.

Parameters

value	<<***out***>> *(*p. *433)* The string into which the middleware should copy the string member. The allocated size of this string is indicated by `size` argument. If the size is sufficient to hold the contents of the member, they will be copied into this string. If `value` is a pointer to NULL, the middleware will allocate a new string for you of sufficient length; it will be your responsibility to free that string. If the size is insufficient but greater than zero, the middleware will *not* free your string; instead, this operation will fail and `size` will contain the minimum required size.
size	<<***inout***>> *(*p. *433)* As an input argument, the allocated size of the string. As an output argument, the actual size of the contents.
member_name	<<***in***>> *(*p. *433)* The name of the member or NULL to look up the member by its ID.
member_id	<<***in***>> *(*p. *433)* The ID of the member or **DDS_DYNAMIC_DATA_MEMBER_ID_UNSPECIFIED** (p. 105) to look up by name. See **Member Names and IDs** (p. 1022).

Exceptions

One	of the **Standard Return Codes** (p. 532). If the member is optional and not set, this operation will return **DDS_RETCODE_NO_DATA** (p. 533). This operation may also return **DDS_RETCODE_OUT_OF_RESOURCES** (p. 533).

See also

DDS_String_alloc (p. 759)

DDS_String_free (p. 762)

DDS_DynamicData::set_wstring (p. 1103)

9.58.6.47 get_int8()

```
DDS_ReturnCode_t DDS_DynamicData::get_int8 (
             DDS_Int8 & value_out,
            const char * member_name,
             DDS_DynamicDataMemberId member_id ) const
```

Get the value of the given field, which is of type **DDS_Int8** (p. 514).

The member may be specified by name or by ID.

This API is not supported when the DynamicData object is in CDR format and will fail with **DDS_RETCODE_↩ PRECONDITION_NOT_MET** (p. 532). See **DDS_DynamicData::is_cdr** (p. 1035) for more information.

MT Safety:

UNSAFE.

Parameters

value_out	*<<**out**>> (p. 433)* If this method returned success, this argument will contain the value of the indicated member.
member_name	*<<**in**>> (p. 433)* The name of the member or NULL to look up the member by its ID.
member_id	*<<**in**>> (p. 433)* The ID of the member or **DDS_DYNAMIC_DATA_MEMBER_ID_UNSPECIFIED** (p. 105) to look up by name. See **Member Names and IDs** (p. 1022).

Exceptions

One	of the **Standard Return Codes** (p. 532). If the member is optional and not set, this operation will return **DDS_RETCODE_NO_DATA** (p. 533)

See also

DDS_DynamicData::set_int8 (p. 1104)

9.58.6.48 get_uint8()

```
DDS_ReturnCode_t DDS_DynamicData::get_uint8 (
             DDS_UInt8 & value_out,
            const char * member_name,
             DDS_DynamicDataMemberId member_id ) const
```

Get the value of the given field, which is of type **DDS_UInt8** (p. 514).

The member may be specified by name or by ID.

This API is not supported when the DynamicData object is in CDR format and will fail with **DDS_RETCODE_↩ PRECONDITION_NOT_MET** (p. 532). See **DDS_DynamicData::is_cdr** (p. 1035) for more information.

MT Safety:

UNSAFE.

Parameters

value_out	*<<**out**>> (p. 433)* If this method returned success, this argument will contain the value of the indicated member.
member_name	*<<**in**>> (p. 433)* The name of the member or NULL to look up the member by its ID.
member_id	*<<**in**>> (p. 433)* The ID of the member or **DDS_DYNAMIC_DATA_MEMBER_ID_UNSPECIFIED** (p. 105) to look up by name. See **Member Names and IDs** (p. 1022).

Exceptions

One	of the **Standard Return Codes** (p. 532). If the member is optional and not set, this operation will return **DDS_RETCODE_NO_DATA** (p. 533)

See also

DDS_DynamicData::set_uint8 (p. 1105)

9.58.6.49 get_complex_member()

```
DDS_ReturnCode_t DDS_DynamicData::get_complex_member (
            DDS_DynamicData & value_out,
           const char * member_name,
            DDS_DynamicDataMemberId member_id ) const
```

Get a copy of the value of the given field, which is of some composed type.

This API is not supported when the DynamicData object is in CDR format and will fail with **DDS_RETCODE_↩ PRECONDITION_NOT_MET** (p. 532). See **DDS_DynamicData::is_cdr** (p. 1035) for more information.

The member may be of type kind **DDS_TK_ARRAY** (p. 89), **DDS_TK_SEQUENCE** (p. 89), **DDS_TK_STRUCT** (p. 89), **DDS_TK_VALUE** (p. 89), or **DDS_TK_UNION** (p. 89). It may be specified by name or by ID.

This method is logically related to **DDS_DynamicData::bind_complex_member** (p. 1042) in that both allow you to examine the state of nested objects. They are different in an important way: this method provides a *copy* of the data; changes to it will not be reflected in the source object.

MT Safety:

UNSAFE.

Parameters

value_out	<<***out***>> *(p. 433)* The **DDS_DynamicData** (p. 1013) sample whose contents will be overwritten by this operation. This object must *not* be a bound member of another **DDS_DynamicData** (p. 1013) sample.
member_name	<<***in***>> *(p. 433)* The name of the member or NULL to look up the member by its ID.
member_id	<<***in***>> *(p. 433)* The ID of the member or **DDS_DYNAMIC_DATA_MEMBER_ID_UNSPECIFIED** (p. 105) to look up by name. See **Member Names and IDs** (p. 1022).

Exceptions

One	of the **Standard Return Codes** (p. 532). If the member is optional and not set, this operation will return **DDS_RETCODE_NO_DATA** (p. 533). This operation may also return **DDS_RETCODE_OUT_OF_RESOURCES** (p. 533).

See also

DDS_DynamicData::set_complex_member (p. 1105)

DDS_DynamicData::bind_complex_member (p. 1042)

9.58.6.50 get_discriminator()

```
DDS_ReturnCode_t DDS_DynamicData::get_discriminator (
            DDS_Long & discriminator ) const
```

Get the current value of the union discriminator.

This operation is valid only for objects of **DDS_TCKind** (p. 89) **DDS_TK_UNION** (p. 89).

This API should be used instead of **DDS_DynamicData::get_member_info_by_index** (p. 1049), which provides the discriminator value for unions when passing in 0 for the index value. That functionality has been deprecated in favor of this API.

This API is not supported when the DynamicData object is in CDR format and will fail with **DDS_RETCODE_↩ PRECONDITION_NOT_MET** (p. 532). See **DDS_DynamicData::is_cdr** (p. 1035) for more information.

MT Safety:

UNSAFE.

Parameters

discriminator	<<***out***>> *(p. 433)* The current value of the union discriminator.

Exceptions

One	of the **Standard Return Codes** (p. 532)

See also

DDS_DynamicData::set_discriminator (p. 1132)

Getting and Setting Union Discriminator Values (p. 1022)

9.58.6.51 get_long_array()

```
DDS_ReturnCode_t DDS_DynamicData::get_long_array (
            DDS_Long * array,
            DDS_UnsignedLong * length,
           const char * member_name,
            DDS_DynamicDataMemberId member_id ) const
```

Get a copy of the given array member. The array may contain members of type **DDS_Long** (p. 515) or **DDS_Enum** (p. 516).

This method will perform an automatic conversion from **DDS_LongSeq** (p. 1192).

If the destination array is insufficiently long to store the data, this operation will fail without copying anything.

This API is not supported when the DynamicData object is in CDR format and will fail with **DDS_RETCODE_↩ PRECONDITION_NOT_MET** (p. 532). See **DDS_DynamicData::is_cdr** (p. 1035) for more information.

MT Safety:

UNSAFE.

Parameters

array	<<***out***>> *(p. 433)* An already-allocated array, into which the elements will be copied.
length	<<***inout***>> *(p. 433)* As an input, the allocated length of `array`. As an output, the number of elements that were copied.
member_name	<<***in***>> *(p. 433)* The name of the member or NULL to look up the member by its ID.
member_id	<<***in***>> *(p. 433)* The ID of the member or **DDS_DYNAMIC_DATA_MEMBER_ID_UNSPECIFIED** (p. 105) to look up by name. See **Member Names and IDs** (p. 1022).

Exceptions

One	of the **Standard Return Codes** (p. 532). If the member is optional and not set, this operation will return **DDS_RETCODE_NO_DATA** (p. 533)

See also

DDS_DynamicData::set_long_array (p. 1107)

DDS_DynamicData::get_long_seq (p. 1081)

9.58.6.52 get_short_array()

```
DDS_ReturnCode_t DDS_DynamicData::get_short_array (
            DDS_Short * array,
```

```
 DDS_UnsignedLong * length,
const char * member_name,
 DDS_DynamicDataMemberId member_id ) const
```

Get a copy of the given array member.

This method will perform an automatic conversion from **DDS_ShortSeq** (p. 1363).

If the destination array is insufficiently long to store the data, this operation will fail without copying anything.

This API is not supported when the DynamicData object is in CDR format and will fail with **DDS_RETCODE_↩ PRECONDITION_NOT_MET** (p. 532). See **DDS_DynamicData::is_cdr** (p. 1035) for more information.

MT Safety:

UNSAFE.

Parameters

array	<<***out***>> *(p. 433)* An already-allocated array, into which the elements will be copied.
length	<<***inout***>> *(p. 433)* As an input, the allocated length of `array`. As an output, the number of elements that were copied.
member_name	<<***in***>> *(p. 433)* The name of the member or NULL to look up the member by its ID.
member_id	<<***in***>> *(p. 433)* The ID of the member or **DDS_DYNAMIC_DATA_MEMBER_ID_UNSPECIFIED** (p. 105) to look up by name. See **Member Names and IDs** (p. 1022).

Exceptions

One	of the **Standard Return Codes** (p. 532). If the member is optional and not set, this operation will return **DDS_RETCODE_NO_DATA** (p. 533)

See also

DDS_DynamicData::set_short_array (p. 1108)

DDS_DynamicData::get_short_seq (p. 1082)

9.58.6.53 get_ulong_array()

```
DDS_ReturnCode_t DDS_DynamicData::get_ulong_array (
             DDS_UnsignedLong * array,
             DDS_UnsignedLong * length,
            const char * member_name,
             DDS_DynamicDataMemberId member_id ) const
```

Get a copy of the given array member. The array may contain members of type **DDS_UnsignedLong** (p. 515) or **DDS_Enum** (p. 516).

This method will perform an automatic conversion from **DDS_UnsignedLongSeq** (p. 1503).

If the destination array is insufficiently long to store the data, this operation will fail without copying anything.

This API is not supported when the DynamicData object is in CDR format and will fail with **DDS_RETCODE_↩ PRECONDITION_NOT_MET** (p. 532). See **DDS_DynamicData::is_cdr** (p. 1035) for more information.

MT Safety:

UNSAFE.

Parameters

array	<<***out***>> *(p. 433)* An already-allocated array, into which the elements will be copied.
length	<<***inout***>> *(p. 433)* As an input, the allocated length of `array`. As an output, the number of elements that were copied.
member_name	<<***in***>> *(p. 433)* The name of the member or NULL to look up the member by its ID.
member_id	<<***in***>> *(p. 433)* The ID of the member or **DDS_DYNAMIC_DATA_MEMBER_ID_UNSPECIFIED** (p. 105) to look up by name. See **Member Names and IDs** (p. 1022).

Exceptions

One	of the **Standard Return Codes** (p. 532). If the member is optional and not set, this operation will return **DDS_RETCODE_NO_DATA** (p. 533)

See also

DDS_DynamicData::set_ulong_array (p. 1109)

DDS_DynamicData::get_ulong_seq (p. 1082)

9.58.6.54 get_ushort_array()

```
DDS_ReturnCode_t DDS_DynamicData::get_ushort_array (
            DDS_UnsignedShort * array,
            DDS_UnsignedLong * length,
           const char * member_name,
            DDS_DynamicDataMemberId member_id ) const
```

Get a copy of the given array member.

This method will perform an automatic conversion from **DDS_UnsignedShortSeq** (p. 1503).

If the destination array is insufficiently long to store the data, this operation will fail without copying anything.

This API is not supported when the DynamicData object is in CDR format and will fail with **DDS_RETCODE_↩ PRECONDITION_NOT_MET** (p. 532). See **DDS_DynamicData::is_cdr** (p. 1035) for more information.

MT Safety:

UNSAFE.

Parameters

array	<<***out***>> *(*p. *433)* An already-allocated array, into which the elements will be copied.
length	<<***inout***>> *(*p. *433)* As an input, the allocated length of `array`. As an output, the number of elements that were copied.
member_name	<<***in***>> *(*p. *433)* The name of the member or NULL to look up the member by its ID.
member_id	<<***in***>> *(*p. *433)* The ID of the member or **DDS_DYNAMIC_DATA_MEMBER_ID_UNSPECIFIED** (p. 105) to look up by name. See **Member Names and IDs** (p. 1022).

Exceptions

One	of the **Standard Return Codes** (p. 532). If the member is optional and not set, this operation will return **DDS_RETCODE_NO_DATA** (p. 533)

See also

DDS_DynamicData::set_ushort_array (p. 1109)

DDS_DynamicData::get_ushort_seq (p. 1083)

9.58.6.55 get_float_array()

```
DDS_ReturnCode_t DDS_DynamicData::get_float_array (
            DDS_Float * array,
            DDS_UnsignedLong * length,
           const char * member_name,
            DDS_DynamicDataMemberId member_id ) const
```

Get a copy of the given array member.

This method will perform an automatic conversion from **DDS_FloatSeq** (p. 1156).

If the destination array is insufficiently long to store the data, this operation will fail without copying anything.

This API is not supported when the DynamicData object is in CDR format and will fail with **DDS_RETCODE_↩ PRECONDITION_NOT_MET** (p. 532). See **DDS_DynamicData::is_cdr** (p. 1035) for more information.

MT Safety:

UNSAFE.

Parameters

array	<<***out***>> *(p. 433)* An already-allocated array, into which the elements will be copied.
length	<<***inout***>> *(p. 433)* As an input, the allocated length of `array`. As an output, the number of elements that were copied.
member_name	<<***in***>> *(p. 433)* The name of the member or NULL to look up the member by its ID.
member_id	<<***in***>> *(p. 433)* The ID of the member or **DDS_DYNAMIC_DATA_MEMBER_ID_UNSPECIFIED** (p. 105) to look up by name. See **Member Names and IDs** (p. 1022).

Exceptions

One	of the **Standard Return Codes** (p. 532). If the member is optional and not set, this operation will return **DDS_RETCODE_NO_DATA** (p. 533)

See also

DDS_DynamicData::set_float_array (p. 1110)

DDS_DynamicData::get_float_seq (p. 1084)

9.58.6.56 get_double_array()

```
DDS_ReturnCode_t DDS_DynamicData::get_double_array (
            DDS_Double * array,
            DDS_UnsignedLong * length,
           const char * member_name,
            DDS_DynamicDataMemberId member_id ) const
```

Get a copy of the given array member.

This method will perform an automatic conversion from **DDS_DoubleSeq** (p. 1004).

If the destination array is insufficiently long to store the data, this operation will fail without copying anything.

This API is not supported when the DynamicData object is in CDR format and will fail with **DDS_RETCODE_↩ PRECONDITION_NOT_MET** (p. 532). See **DDS_DynamicData::is_cdr** (p. 1035) for more information.

MT Safety:

UNSAFE.

Parameters

array	<<***out***>> *(p. 433)* An already-allocated array, into which the elements will be copied.
length	<<***inout***>> *(p. 433)* As an input, the allocated length of `array`. As an output, the number of elements that were copied.
member_name	<<***in***>> *(p. 433)* The name of the member or NULL to look up the member by its ID.
member_id	<<***in***>> *(p. 433)* The ID of the member or **DDS_DYNAMIC_DATA_MEMBER_ID_UNSPECIFIED** (p. 105) to look up by name. See **Member Names and IDs** (p. 1022).

Exceptions

One	of the **Standard Return Codes** (p. 532). If the member is optional and not set, this operation will return **DDS_RETCODE_NO_DATA** (p. 533)

See also

DDS_DynamicData::set_double_array (p. 1111)

DDS_DynamicData::get_double_seq (p. 1085)

9.58.6.57 get_boolean_array()

```
DDS_ReturnCode_t DDS_DynamicData::get_boolean_array (
            DDS_Boolean * array,
            DDS_UnsignedLong * length,
           const char * member_name,
            DDS_DynamicDataMemberId member_id ) const
```

Get a copy of the given array member.

This method will perform an automatic conversion from **DDS_BooleanSeq** (p. 814).

If the destination array is insufficiently long to store the data, this operation will fail without copying anything.

This API is not supported when the DynamicData object is in CDR format and will fail with **DDS_RETCODE_↩ PRECONDITION_NOT_MET** (p. 532). See **DDS_DynamicData::is_cdr** (p. 1035) for more information.

MT Safety:

UNSAFE.

Parameters

array	<<***out***>> *(p. 433)* An already-allocated array, into which the elements will be copied.
length	<<***inout***>> *(p. 433)* As an input, the allocated length of `array`. As an output, the number of elements that were copied.
member_name	<<***in***>> *(p. 433)* The name of the member or NULL to look up the member by its ID.
member_id	<<***in***>> *(p. 433)* The ID of the member or **DDS_DYNAMIC_DATA_MEMBER_ID_UNSPECIFIED** (p. 105) to look up by name. See **Member Names and IDs** (p. 1022).

Exceptions

One	of the **Standard Return Codes** (p. 532). If the member is optional and not set, this operation will return **DDS_RETCODE_NO_DATA** (p. 533)

See also

DDS_DynamicData::set_boolean_array (p. 1112)

DDS_DynamicData::get_boolean_seq (p. 1086)

9.58.6.58 get_char_array()

```
DDS_ReturnCode_t DDS_DynamicData::get_char_array (
            DDS_Char * array,
            DDS_UnsignedLong * length,
           const char * member_name,
            DDS_DynamicDataMemberId member_id ) const
```

Get a copy of the given array member.

This method will perform an automatic conversion from **DDS_CharSeq** (p. 824).

If the destination array is insufficiently long to store the data, this operation will fail without copying anything.

This API is not supported when the DynamicData object is in CDR format and will fail with **DDS_RETCODE_↩ PRECONDITION_NOT_MET** (p. 532). See **DDS_DynamicData::is_cdr** (p. 1035) for more information.

MT Safety:

UNSAFE.

Parameters

array	<<***out***>> *(p. 433)* An already-allocated array, into which the elements will be copied.
length	<<***inout***>> *(p. 433)* As an input, the allocated length of `array`. As an output, the number of elements that were copied.
member_name	<<***in***>> *(p. 433)* The name of the member or NULL to look up the member by its ID.
member_id	<<***in***>> *(p. 433)* The ID of the member or **DDS_DYNAMIC_DATA_MEMBER_ID_UNSPECIFIED** (p. 105) to look up by name. See **Member Names and IDs** (p. 1022).

Exceptions

One	of the **Standard Return Codes** (p. 532). If the member is optional and not set, this operation will return **DDS_RETCODE_NO_DATA** (p. 533)

See also

DDS_DynamicData::set_char_array (p. 1113)

DDS_DynamicData::get_char_seq (p. 1087)

9.58.6.59 get_octet_array()

```
DDS_ReturnCode_t DDS_DynamicData::get_octet_array (
            DDS_Octet * array,
            DDS_UnsignedLong * length,
           const char * member_name,
            DDS_DynamicDataMemberId member_id ) const
```

Get a copy of the given array member.

This method will perform an automatic conversion from **DDS_OctetSeq** (p. 1218).

If the destination array is insufficiently long to store the data, this operation will fail without copying anything.

This API is not supported when the DynamicData object is in CDR format and will fail with **DDS_RETCODE_↩ PRECONDITION_NOT_MET** (p. 532). See **DDS_DynamicData::is_cdr** (p. 1035) for more information.

MT Safety:

UNSAFE.

Parameters

array	<<***out***>> *(p. 433)* An already-allocated array, into which the elements will be copied.
length	<<***inout***>> *(p. 433)* As an input, the allocated length of `array`. As an output, the number of elements that were copied.
member_name	<<***in***>> *(p. 433)* The name of the member or NULL to look up the member by its ID.
member_id	<<***in***>> *(p. 433)* The ID of the member or **DDS_DYNAMIC_DATA_MEMBER_ID_UNSPECIFIED** (p. 105) to look up by name. See **Member Names and IDs** (p. 1022).

Exceptions

One	of the **Standard Return Codes** (p. 532). If the member is optional and not set, this operation will return **DDS_RETCODE_NO_DATA** (p. 533)

See also

DDS_DynamicData::set_octet_array (p. 1114)

DDS_DynamicData::get_octet_seq (p. 1087)

9.58.6.60 get_longlong_array()

```
DDS_ReturnCode_t DDS_DynamicData::get_longlong_array (
            DDS_LongLong * array,
            DDS_UnsignedLong * length,
           const char * member_name,
            DDS_DynamicDataMemberId member_id ) const
```

Get a copy of the given array member.

This method will perform an automatic conversion from **DDS_LongLongSeq** (p. 1192).

If the destination array is insufficiently long to store the data, this operation will fail without copying anything.

This API is not supported when the DynamicData object is in CDR format and will fail with **DDS_RETCODE_↩ PRECONDITION_NOT_MET** (p. 532). See **DDS_DynamicData::is_cdr** (p. 1035) for more information.

MT Safety:

UNSAFE.

Parameters

array	<<***out***>> *(p. 433)* An already-allocated array, into which the elements will be copied.
length	<<***inout***>> *(p. 433)* As an input, the allocated length of `array`. As an output, the number of elements that were copied.
member_name	<<***in***>> *(p. 433)* The name of the member or NULL to look up the member by its ID.
member_id	<<***in***>> *(p. 433)* The ID of the member or **DDS_DYNAMIC_DATA_MEMBER_ID_UNSPECIFIED** (p. 105) to look up by name. See **Member Names and IDs** (p. 1022).

Exceptions

One	of the **Standard Return Codes** (p. 532). If the member is optional and not set, this operation will return **DDS_RETCODE_NO_DATA** (p. 533)

See also

DDS_DynamicData::set_longlong_array (p. 1115)

DDS_DynamicData::get_longlong_seq (p. 1088)

9.58.6.61 get_ulonglong_array()

```
DDS_ReturnCode_t DDS_DynamicData::get_ulonglong_array (
            DDS_UnsignedLongLong * array,
```

```
 DDS_UnsignedLong * length,
const char * member_name,
 DDS_DynamicDataMemberId member_id ) const
```

Get a copy of the given array member.

This method will perform an automatic conversion from **DDS_UnsignedLongLongSeq** (p. 1503).

If the destination array is insufficiently long to store the data, this operation will fail without copying anything.

This API is not supported when the DynamicData object is in CDR format and will fail with **DDS_RETCODE_↩ PRECONDITION_NOT_MET** (p. 532). See **DDS_DynamicData::is_cdr** (p. 1035) for more information.

MT Safety:

UNSAFE.

Parameters

array	<<***out***>> *(p. 433)* An already-allocated array, into which the elements will be copied.
length	<<***inout***>> *(p. 433)* As an input, the allocated length of `array`. As an output, the number of elements that were copied.
member_name	<<***in***>> *(p. 433)* The name of the member or NULL to look up the member by its ID.
member_id	<<***in***>> *(p. 433)* The ID of the member or **DDS_DYNAMIC_DATA_MEMBER_ID_UNSPECIFIED** (p. 105) to look up by name. See **Member Names and IDs** (p. 1022).

Exceptions

One	of the **Standard Return Codes** (p. 532). If the member is optional and not set, this operation will return **DDS_RETCODE_NO_DATA** (p. 533)

See also

DDS_DynamicData::set_ulonglong_array (p. 1115)

DDS_DynamicData::get_ulonglong_seq (p. 1089)

9.58.6.62 get_longdouble_array()

```
DDS_ReturnCode_t DDS_DynamicData::get_longdouble_array (
             DDS_LongDouble * array,
             DDS_UnsignedLong * length,
            const char * member_name,
             DDS_DynamicDataMemberId member_id ) const
```

Get a copy of the given array member.

This method will perform an automatic conversion from **DDS_LongDoubleSeq** (p. 1192).

If the destination array is insufficiently long to store the data, this operation will fail without copying anything.

This API is not supported when the DynamicData object is in CDR format and will fail with **DDS_RETCODE_↩ PRECONDITION_NOT_MET** (p. 532). See **DDS_DynamicData::is_cdr** (p. 1035) for more information.

MT Safety:

UNSAFE.

Parameters

array	<<***out***>> *(p. 433)* An already-allocated array, into which the elements will be copied.
length	<<***inout***>> *(p. 433)* As an input, the allocated length of `array`. As an output, the number of elements that were copied.
member_name	<<***in***>> *(p. 433)* The name of the member or NULL to look up the member by its ID.
member_id	<<***in***>> *(p. 433)* The ID of the member or **DDS_DYNAMIC_DATA_MEMBER_ID_UNSPECIFIED** (p. 105) to look up by name. See **Member Names and IDs** (p. 1022).

Exceptions

One	of the **Standard Return Codes** (p. 532). If the member is optional and not set, this operation will return **DDS_RETCODE_NO_DATA** (p. 533)

See also

DDS_DynamicData::set_longdouble_array (p. 1116)

DDS_DynamicData::get_longdouble_seq (p. 1090)

9.58.6.63 get_wchar_array()

```
DDS_ReturnCode_t DDS_DynamicData::get_wchar_array (
            DDS_Wchar * array,
            DDS_UnsignedLong * length,
           const char * member_name,
            DDS_DynamicDataMemberId member_id ) const
```

Get a copy of the given array member.

This method will perform an automatic conversion from **DDS_WcharSeq** (p. 1510).

If the destination array is insufficiently long to store the data, this operation will fail without copying anything.

This API is not supported when the DynamicData object is in CDR format and will fail with **DDS_RETCODE_↩ PRECONDITION_NOT_MET** (p. 532). See **DDS_DynamicData::is_cdr** (p. 1035) for more information.

MT Safety:

UNSAFE.

Parameters

array	<<***out***>> *(*p. *433)* An already-allocated array, into which the elements will be copied.
length	<<***inout***>> *(*p. *433)* As an input, the allocated length of `array`. As an output, the number of elements that were copied.
member_name	<<***in***>> *(*p. *433)* The name of the member or NULL to look up the member by its ID.
member_id	<<***in***>> *(*p. *433)* The ID of the member or **DDS_DYNAMIC_DATA_MEMBER_ID_UNSPECIFIED** (p. 105) to look up by name. See **Member Names and IDs** (p. 1022).

Exceptions

One	of the **Standard Return Codes** (p. 532). If the member is optional and not set, this operation will return **DDS_RETCODE_NO_DATA** (p. 533)

See also

DDS_DynamicData::set_wchar_array (p. 1117)

DDS_DynamicData::get_wchar_seq (p. 1091)

9.58.6.64 get_int8_array()

```
DDS_ReturnCode_t DDS_DynamicData::get_int8_array (
            DDS_Int8 * array,
            DDS_UnsignedLong * length,
           const char * member_name,
            DDS_DynamicDataMemberId member_id ) const
```

Get a copy of the given array member.

This method will perform an automatic conversion from **DDS_Int8Seq** (p. 1167).

If the destination array is insufficiently long to store the data, this operation will fail without copying anything.

This API is not supported when the DynamicData object is in CDR format and will fail with **DDS_RETCODE_↩ PRECONDITION_NOT_MET** (p. 532). See **DDS_DynamicData::is_cdr** (p. 1035) for more information.

MT Safety:

UNSAFE.

Parameters

array	<<***out***>> *(*p. *433)* An already-allocated array, into which the elements will be copied.
length	<<***inout***>> *(*p. *433)* As an input, the allocated length of `array`. As an output, the number of elements that were copied.
member_name	<<***in***>> *(*p. *433)* The name of the member or NULL to look up the member by its ID.
member_id	<<***in***>> *(*p. *433)* The ID of the member or **DDS_DYNAMIC_DATA_MEMBER_ID_UNSPECIFIED** (p. 105) to look up by name. See **Member Names and IDs** (p. 1022).

Exceptions

One	of the **Standard Return Codes** (p. 532). If the member is optional and not set, this operation will return **DDS_RETCODE_NO_DATA** (p. 533)

See also

DDS_DynamicData::set_int8_array (p. 1118)

DDS_DynamicData::get_int8_seq (p. 1092)

9.58.6.65 get_uint8_array()

```
DDS_ReturnCode_t DDS_DynamicData::get_uint8_array (
            DDS_UInt8 * array,
            DDS_UnsignedLong * length,
           const char * member_name,
            DDS_DynamicDataMemberId member_id ) const
```

Get a copy of the given array member.

This method will perform an automatic conversion from **DDS_UInt8Seq** (p. 1500).

If the destination array is insufficiently long to store the data, this operation will fail without copying anything.

This API is not supported when the DynamicData object is in CDR format and will fail with **DDS_RETCODE_↩ PRECONDITION_NOT_MET** (p. 532). See **DDS_DynamicData::is_cdr** (p. 1035) for more information.

MT Safety:

UNSAFE.

Parameters

array	<<***out***>> *(*p. *433)* An already-allocated array, into which the elements will be copied.
length	<<***inout***>> *(*p. *433)* As an input, the allocated length of `array`. As an output, the number of elements that were copied.
member_name	<<***in***>> *(*p. *433)* The name of the member or NULL to look up the member by its ID.
member_id	<<***in***>> *(*p. *433)* The ID of the member or **DDS_DYNAMIC_DATA_MEMBER_ID_UNSPECIFIED** (p. 105) to look up by name. See **Member Names and IDs** (p. 1022).

Exceptions

One	of the **Standard Return Codes** (p. 532). If the member is optional and not set, this operation will return **DDS_RETCODE_NO_DATA** (p. 533)

See also

DDS_DynamicData::set_uint8_array (p. 1119)

DDS_DynamicData::get_uint8_seq (p. 1092)

9.58.6.66 get_long_seq()

```
DDS_ReturnCode_t DDS_DynamicData::get_long_seq (
            DDS_LongSeq & seq,
           const char * member_name,
            DDS_DynamicDataMemberId member_id ) const
```

Get a copy of the given sequence member.

The provided sequence will be automatically resized as necessary.

This method will perform an automatic conversion from an array of **DDS_Long** (p. 515).

This API is not supported when the DynamicData object is in CDR format and will fail with **DDS_RETCODE_↩ PRECONDITION_NOT_MET** (p. 532). See **DDS_DynamicData::is_cdr** (p. 1035) for more information.

MT Safety:

UNSAFE.

Parameters

seq	<<***out***>> *(p. 433)* A sequence, into which the elements will be copied.
member_name	<<***in***>> *(p. 433)* The name of the member or NULL to look up the member by its ID.
member_id	<<***in***>> *(p. 433)* The ID of the member or **DDS_DYNAMIC_DATA_MEMBER_ID_UNSPECIFIED** (p. 105) to look up by name. See **Member Names and IDs** (p. 1022).

Exceptions

One	of the **Standard Return Codes** (p. 532). If the member is optional and not set, this operation will return **DDS_RETCODE_NO_DATA** (p. 533). This operation may also return **DDS_RETCODE_OUT_OF_RESOURCES** (p. 533).

See also

DDS_DynamicData::set_long_seq (p. 1120)

DDS_DynamicData::get_long_array (p. 1067)

9.58.6.67 get_short_seq()

```
DDS_ReturnCode_t DDS_DynamicData::get_short_seq (
            DDS_ShortSeq & seq,
           const char * member_name,
            DDS_DynamicDataMemberId member_id ) const
```

Get a copy of the given sequence member.

The provided sequence will be automatically resized as necessary.

This method will perform an automatic conversion from an array of **DDS_Short** (p. 514).

This API is not supported when the DynamicData object is in CDR format and will fail with **DDS_RETCODE_↩ PRECONDITION_NOT_MET** (p. 532). See **DDS_DynamicData::is_cdr** (p. 1035) for more information.

MT Safety:

UNSAFE.

Parameters

seq	<<***out***>> *(p. 433)* A sequence, into which the elements will be copied.
member_name	<<***in***>> *(p. 433)* The name of the member or NULL to look up the member by its ID.
member_id	<<***in***>> *(p. 433)* The ID of the member or **DDS_DYNAMIC_DATA_MEMBER_ID_UNSPECIFIED** (p. 105) to look up by name. See **Member Names and IDs** (p. 1022).

Exceptions

One	of the **Standard Return Codes** (p. 532). If the member is optional and not set, this operation will return **DDS_RETCODE_NO_DATA** (p. 533). This operation may also return **DDS_RETCODE_OUT_OF_RESOURCES** (p. 533).

See also

DDS_DynamicData::set_short_seq (p. 1121)

DDS_DynamicData::get_short_array (p. 1068)

9.58.6.68 get_ulong_seq()

```
DDS_ReturnCode_t DDS_DynamicData::get_ulong_seq (
            DDS_UnsignedLongSeq & seq,
           const char * member_name,
            DDS_DynamicDataMemberId member_id ) const
```

Get a copy of the given sequence member.

The provided sequence will be automatically resized as necessary.

This method will perform an automatic conversion from an array of **DDS_UnsignedLong** (p. 515).

This API is not supported when the DynamicData object is in CDR format and will fail with **DDS_RETCODE_↩ PRECONDITION_NOT_MET** (p. 532). See **DDS_DynamicData::is_cdr** (p. 1035) for more information.

MT Safety:

UNSAFE.

Parameters

seq	<<***out***>> *(p. 433)* A sequence, into which the elements will be copied.
member_name	<<***in***>> *(p. 433)* The name of the member or NULL to look up the member by its ID.
member_id	<<***in***>> *(p. 433)* The ID of the member or **DDS_DYNAMIC_DATA_MEMBER_ID_UNSPECIFIED** (p. 105) to look up by name. See **Member Names and IDs** (p. 1022).

Exceptions

One	of the **Standard Return Codes** (p. 532). If the member is optional and not set, this operation will return **DDS_RETCODE_NO_DATA** (p. 533). This operation may also return **DDS_RETCODE_OUT_OF_RESOURCES** (p. 533).

See also

DDS_DynamicData::set_ulong_seq (p. 1121)

DDS_DynamicData::get_ulong_array (p. 1069)

9.58.6.69 get_ushort_seq()

```
DDS_ReturnCode_t DDS_DynamicData::get_ushort_seq (
            DDS_UnsignedShortSeq & seq,
           const char * member_name,
            DDS_DynamicDataMemberId member_id ) const
```

Get a copy of the given sequence member.

The provided sequence will be automatically resized as necessary.

This method will perform an automatic conversion from an array of **DDS_UnsignedShort** (p. 514).

This API is not supported when the DynamicData object is in CDR format and will fail with **DDS_RETCODE_↩ PRECONDITION_NOT_MET** (p. 532). See **DDS_DynamicData::is_cdr** (p. 1035) for more information.

MT Safety:

UNSAFE.

Parameters

seq	<<***out***>> *(p. 433)* A sequence, into which the elements will be copied.
member_name	<<***in***>> *(p. 433)* The name of the member or NULL to look up the member by its ID.
member_id	<<***in***>> *(p. 433)* The ID of the member or **DDS_DYNAMIC_DATA_MEMBER_ID_UNSPECIFIED** (p. 105) to look up by name. See **Member Names and IDs** (p. 1022).

Exceptions

One	of the **Standard Return Codes** (p. 532). If the member is optional and not set, this operation will return **DDS_RETCODE_NO_DATA** (p. 533). This operation may also return **DDS_RETCODE_OUT_OF_RESOURCES** (p. 533).

See also

DDS_DynamicData::set_ushort_seq (p. 1122)

DDS_DynamicData::get_ushort_array (p. 1070)

9.58.6.70 get_float_seq()

```
DDS_ReturnCode_t DDS_DynamicData::get_float_seq (
            DDS_FloatSeq & seq,
           const char * member_name,
            DDS_DynamicDataMemberId member_id ) const
```

Get a copy of the given sequence member.

The provided sequence will be automatically resized as necessary.

This method will perform an automatic conversion from an array of **DDS_Long** (p. 515).

This API is not supported when the DynamicData object is in CDR format and will fail with **DDS_RETCODE_↩ PRECONDITION_NOT_MET** (p. 532). See **DDS_DynamicData::is_cdr** (p. 1035) for more information.

MT Safety:

UNSAFE.

Parameters

seq	<<***out***>> *(p. 433)* A sequence, into which the elements will be copied.
member_name	<<***in***>> *(p. 433)* The name of the member or NULL to look up the member by its ID.
member_id	<<***in***>> *(p. 433)* The ID of the member or **DDS_DYNAMIC_DATA_MEMBER_ID_UNSPECIFIED** (p. 105) to look up by name. See **Member Names and IDs** (p. 1022).

Exceptions

One	of the **Standard Return Codes** (p. 532). If the member is optional and not set, this operation will return **DDS_RETCODE_NO_DATA** (p. 533). This operation may also return **DDS_RETCODE_OUT_OF_RESOURCES** (p. 533).

See also

DDS_DynamicData::set_long_seq (p. 1120)

DDS_DynamicData::get_long_array (p. 1067)

9.58.6.71 get_double_seq()

```
DDS_ReturnCode_t DDS_DynamicData::get_double_seq (
            DDS_DoubleSeq & seq,
           const char * member_name,
            DDS_DynamicDataMemberId member_id ) const
```

Get a copy of the given sequence member.

The provided sequence will be automatically resized as necessary.

This method will perform an automatic conversion from an array of **DDS_Double** (p. 515).

This API is not supported when the DynamicData object is in CDR format and will fail with **DDS_RETCODE_↩ PRECONDITION_NOT_MET** (p. 532). See **DDS_DynamicData::is_cdr** (p. 1035) for more information.

MT Safety:

UNSAFE.

Parameters

seq	<<***out***>> *(p. 433)* A sequence, into which the elements will be copied.
member_name	<<***in***>> *(p. 433)* The name of the member or NULL to look up the member by its ID.
member_id	<<***in***>> *(p. 433)* The ID of the member or **DDS_DYNAMIC_DATA_MEMBER_ID_UNSPECIFIED** (p. 105) to look up by name. See **Member Names and IDs** (p. 1022).

Exceptions

One	of the **Standard Return Codes** (p. 532). If the member is optional and not set, this operation will return **DDS_RETCODE_NO_DATA** (p. 533). This operation may also return **DDS_RETCODE_OUT_OF_RESOURCES** (p. 533).

See also

DDS_DynamicData::set_double_seq (p. 1124)

DDS_DynamicData::get_double_array (p. 1072)

9.58.6.72 get_boolean_seq()

```
DDS_ReturnCode_t DDS_DynamicData::get_boolean_seq (
            DDS_BooleanSeq & seq,
           const char * member_name,
            DDS_DynamicDataMemberId member_id ) const
```

Get a copy of the given sequence member.

The provided sequence will be automatically resized as necessary.

This method will perform an automatic conversion from an array of **DDS_Boolean** (p. 516).

This API is not supported when the DynamicData object is in CDR format and will fail with **DDS_RETCODE_↩ PRECONDITION_NOT_MET** (p. 532). See **DDS_DynamicData::is_cdr** (p. 1035) for more information.

MT Safety:

UNSAFE.

Parameters

seq	<<***out***>> *(p. 433)* A sequence, into which the elements will be copied.
member_name	<<***in***>> *(p. 433)* The name of the member or NULL to look up the member by its ID.
member_id	<<***in***>> *(p. 433)* The ID of the member or **DDS_DYNAMIC_DATA_MEMBER_ID_UNSPECIFIED** (p. 105) to look up by name. See **Member Names and IDs** (p. 1022).

Exceptions

One	of the **Standard Return Codes** (p. 532). If the member is optional and not set, this operation will return **DDS_RETCODE_NO_DATA** (p. 533). This operation may also return **DDS_RETCODE_OUT_OF_RESOURCES** (p. 533).

See also

DDS_DynamicData::set_boolean_seq (p. 1125)

DDS_DynamicData::get_boolean_array (p. 1073)

9.58.6.73 get_char_seq()

```
DDS_ReturnCode_t DDS_DynamicData::get_char_seq (
            DDS_CharSeq & seq,
           const char * member_name,
            DDS_DynamicDataMemberId member_id ) const
```

Get a copy of the given sequence member.

The provided sequence will be automatically resized as necessary.

This method will perform an automatic conversion from an array of **DDS_Char** (p. 513).

This API is not supported when the DynamicData object is in CDR format and will fail with **DDS_RETCODE_↩ PRECONDITION_NOT_MET** (p. 532). See **DDS_DynamicData::is_cdr** (p. 1035) for more information.

MT Safety:

UNSAFE.

Parameters

seq	<<***out***>> *(p. 433)* A sequence, into which the elements will be copied.
member_name	<<***in***>> *(p. 433)* The name of the member or NULL to look up the member by its ID.
member_id	<<***in***>> *(p. 433)* The ID of the member or **DDS_DYNAMIC_DATA_MEMBER_ID_UNSPECIFIED** (p. 105) to look up by name. See **Member Names and IDs** (p. 1022).

Exceptions

One	of the **Standard Return Codes** (p. 532). If the member is optional and not set, this operation will return **DDS_RETCODE_NO_DATA** (p. 533). This operation may also return **DDS_RETCODE_OUT_OF_RESOURCES** (p. 533).

See also

DDS_DynamicData::set_char_seq (p. 1125)

DDS_DynamicData::get_char_array (p. 1074)

9.58.6.74 get_octet_seq()

```
DDS_ReturnCode_t DDS_DynamicData::get_octet_seq (
            DDS_OctetSeq & seq,
           const char * member_name,
            DDS_DynamicDataMemberId member_id ) const
```

Get a copy of the given sequence member.

The provided sequence will be automatically resized as necessary.

This method will perform an automatic conversion from an array of **DDS_Octet** (p. 514).

This API is not supported when the DynamicData object is in CDR format and will fail with **DDS_RETCODE_↩ PRECONDITION_NOT_MET** (p. 532). See **DDS_DynamicData::is_cdr** (p. 1035) for more information.

MT Safety:

UNSAFE.

Parameters

seq	<<***out***>> *(p. 433)* A sequence, into which the elements will be copied.
member_name	<<***in***>> *(p. 433)* The name of the member or NULL to look up the member by its ID.
member_id	<<***in***>> *(p. 433)* The ID of the member or **DDS_DYNAMIC_DATA_MEMBER_ID_UNSPECIFIED** (p. 105) to look up by name. See **Member Names and IDs** (p. 1022).

Exceptions

One	of the **Standard Return Codes** (p. 532). If the member is optional and not set, this operation will return **DDS_RETCODE_NO_DATA** (p. 533). This operation may also return **DDS_RETCODE_OUT_OF_RESOURCES** (p. 533).

See also

DDS_DynamicData::set_octet_seq (p. 1126)

DDS_DynamicData::get_octet_array (p. 1075)

9.58.6.75 get_longlong_seq()

```
DDS_ReturnCode_t DDS_DynamicData::get_longlong_seq (
            DDS_LongLongSeq & seq,
           const char * member_name,
            DDS_DynamicDataMemberId member_id ) const
```

Get a copy of the given sequence member.

The provided sequence will be automatically resized as necessary.

This method will perform an automatic conversion from an array of **DDS_LongLong** (p. 515).

This API is not supported when the DynamicData object is in CDR format and will fail with **DDS_RETCODE_↩ PRECONDITION_NOT_MET** (p. 532). See **DDS_DynamicData::is_cdr** (p. 1035) for more information.

MT Safety:

UNSAFE.

Parameters

seq	<<***out***>> *(p. 433)* A sequence, into which the elements will be copied.
member_name	<<***in***>> *(p. 433)* The name of the member or NULL to look up the member by its ID.
member_id	<<***in***>> *(p. 433)* The ID of the member or **DDS_DYNAMIC_DATA_MEMBER_ID_UNSPECIFIED** (p. 105) to look up by name. See **Member Names and IDs** (p. 1022).

Exceptions

One	of the **Standard Return Codes** (p. 532). If the member is optional and not set, this operation will return **DDS_RETCODE_NO_DATA** (p. 533). This operation may also return **DDS_RETCODE_OUT_OF_RESOURCES** (p. 533).

See also

DDS_DynamicData::set_longlong_seq (p. 1127)

DDS_DynamicData::get_longlong_array (p. 1075)

9.58.6.76 get_ulonglong_seq()

```
DDS_ReturnCode_t DDS_DynamicData::get_ulonglong_seq (
            DDS_UnsignedLongLongSeq & seq,
           const char * member_name,
            DDS_DynamicDataMemberId member_id ) const
```

Get a copy of the given sequence member.

The provided sequence will be automatically resized as necessary.

This method will perform an automatic conversion from an array of **DDS_UnsignedLongLong** (p. 515).

This API is not supported when the DynamicData object is in CDR format and will fail with **DDS_RETCODE_↩ PRECONDITION_NOT_MET** (p. 532). See **DDS_DynamicData::is_cdr** (p. 1035) for more information.

MT Safety:

UNSAFE.

Parameters

seq	<<***out***>> *(p. 433)* A sequence, into which the elements will be copied.
member_name	<<***in***>> *(p. 433)* The name of the member or NULL to look up the member by its ID.
member_id	<<***in***>> *(p. 433)* The ID of the member or **DDS_DYNAMIC_DATA_MEMBER_ID_UNSPECIFIED** (p. 105) to look up by name. See **Member Names and IDs** (p. 1022).

Exceptions

One	of the **Standard Return Codes** (p. 532). If the member is optional and not set, this operation will return **DDS_RETCODE_NO_DATA** (p. 533). This operation may also return **DDS_RETCODE_OUT_OF_RESOURCES** (p. 533).

See also

DDS_DynamicData::set_ulonglong_seq (p. 1128)

DDS_DynamicData::get_ulonglong_array (p. 1076)

9.58.6.77 get_longdouble_seq()

```
DDS_ReturnCode_t DDS_DynamicData::get_longdouble_seq (
            DDS_LongDoubleSeq & seq,
           const char * member_name,
            DDS_DynamicDataMemberId member_id ) const
```

Get a copy of the given sequence member.

The provided sequence will be automatically resized as necessary.

This method will perform an automatic conversion from an array of **DDS_LongDouble** (p. 516).

This API is not supported when the DynamicData object is in CDR format and will fail with **DDS_RETCODE_↩ PRECONDITION_NOT_MET** (p. 532). See **DDS_DynamicData::is_cdr** (p. 1035) for more information.

MT Safety:

UNSAFE.

Parameters

seq	<<***out***>> *(p. 433)* A sequence, into which the elements will be copied.
member_name	<<***in***>> *(p. 433)* The name of the member or NULL to look up the member by its ID.
member_id	<<***in***>> *(p. 433)* The ID of the member or **DDS_DYNAMIC_DATA_MEMBER_ID_UNSPECIFIED** (p. 105) to look up by name. See **Member Names and IDs** (p. 1022).

Exceptions

One	of the **Standard Return Codes** (p. 532). If the member is optional and not set, this operation will return **DDS_RETCODE_NO_DATA** (p. 533). This operation may also return **DDS_RETCODE_OUT_OF_RESOURCES** (p. 533).

See also

DDS_DynamicData::set_longdouble_seq (p. 1129)

DDS_DynamicData::get_longdouble_array (p. 1077)

9.58.6.78 get_wchar_seq()

```
DDS_ReturnCode_t DDS_DynamicData::get_wchar_seq (
            DDS_WcharSeq & seq,
           const char * member_name,
            DDS_DynamicDataMemberId member_id ) const
```

Get a copy of the given sequence member.

The provided sequence will be automatically resized as necessary.

This method will perform an automatic conversion from an array of **DDS_Wchar** (p. 514).

This API is not supported when the DynamicData object is in CDR format and will fail with **DDS_RETCODE_↩ PRECONDITION_NOT_MET** (p. 532). See **DDS_DynamicData::is_cdr** (p. 1035) for more information.

MT Safety:

UNSAFE.

Parameters

seq	<<***out***>> *(p. 433)* A sequence, into which the elements will be copied.
member_name	<<***in***>> *(p. 433)* The name of the member or NULL to look up the member by its ID.
member_id	<<***in***>> *(p. 433)* The ID of the member or **DDS_DYNAMIC_DATA_MEMBER_ID_UNSPECIFIED** (p. 105) to look up by name. See **Member Names and IDs** (p. 1022).

Exceptions

One	of the **Standard Return Codes** (p. 532). If the member is optional and not set, this operation will return **DDS_RETCODE_NO_DATA** (p. 533). This operation may also return **DDS_RETCODE_OUT_OF_RESOURCES** (p. 533).

See also

DDS_DynamicData::set_wchar_seq (p. 1129)

DDS_DynamicData::get_wchar_array (p. 1078)

9.58.6.79 get_int8_seq()

```
DDS_ReturnCode_t DDS_DynamicData::get_int8_seq (
            DDS_Int8Seq & seq,
           const char * member_name,
            DDS_DynamicDataMemberId member_id ) const
```

Get a copy of the given sequence member.

The provided sequence will be automatically resized as necessary.

This method will perform an automatic conversion from an array of **DDS_Int8** (p. 514).

This API is not supported when the DynamicData object is in CDR format and will fail with **DDS_RETCODE_↩ PRECONDITION_NOT_MET** (p. 532). See **DDS_DynamicData::is_cdr** (p. 1035) for more information.

MT Safety:

UNSAFE.

Parameters

seq	<<***out***>> *(p. 433)* A sequence, into which the elements will be copied.
member_name	<<***in***>> *(p. 433)* The name of the member or NULL to look up the member by its ID.
member_id	<<***in***>> *(p. 433)* The ID of the member or **DDS_DYNAMIC_DATA_MEMBER_ID_UNSPECIFIED** (p. 105) to look up by name. See **Member Names and IDs** (p. 1022).

Exceptions

One	of the **Standard Return Codes** (p. 532). If the member is optional and not set, this operation will return **DDS_RETCODE_NO_DATA** (p. 533). This operation may also return **DDS_RETCODE_OUT_OF_RESOURCES** (p. 533).

See also

DDS_DynamicData::set_int8_seq (p. 1130)

DDS_DynamicData::get_int8_array (p. 1079)

9.58.6.80 get_uint8_seq()

```
DDS_ReturnCode_t DDS_DynamicData::get_uint8_seq (
            DDS_UInt8Seq & seq,
            const char * member_name,
            DDS_DynamicDataMemberId member_id ) const
```

Get a copy of the given sequence member.

The provided sequence will be automatically resized as necessary.

This method will perform an automatic conversion from an array of **DDS_UInt8** (p. 514).

This API is not supported when the DynamicData object is in CDR format and will fail with **DDS_RETCODE_↩ PRECONDITION_NOT_MET** (p. 532). See **DDS_DynamicData::is_cdr** (p. 1035) for more information.

MT Safety:

UNSAFE.

Parameters

seq	<<***out***>> *(p. 433)* A sequence, into which the elements will be copied.
member_name	<<***in***>> *(p. 433)* The name of the member or NULL to look up the member by its ID.
member_id	<<***in***>> *(p. 433)* The ID of the member or **DDS_DYNAMIC_DATA_MEMBER_ID_UNSPECIFIED** (p. 105) to look up by name. See **Member Names and IDs** (p. 1022).

Exceptions

One	of the **Standard Return Codes** (p. 532). If the member is optional and not set, this operation will return **DDS_RETCODE_NO_DATA** (p. 533). This operation may also return **DDS_RETCODE_OUT_OF_RESOURCES** (p. 533).

See also

DDS_DynamicData::set_uint8_seq (p. 1131)

DDS_DynamicData::get_uint8_array (p. 1080)

9.58.6.81 set_long()

```
DDS_ReturnCode_t DDS_DynamicData::set_long (
            const char * member_name,
            DDS_DynamicDataMemberId member_id,
            DDS_Long value )
```

Set the value of the given field, which is of type **DDS_Long** (p. 515).

This API is not supported when the DynamicData object is in CDR format and will fail with **DDS_RETCODE_↩ PRECONDITION_NOT_MET** (p. 532). See **DDS_DynamicData::is_cdr** (p. 1035) for more information.

MT Safety:

UNSAFE.

Parameters

member_name	*<<**in**>> (p. 433)* The name of the member or NULL to look up the member by its ID.
member_id	*<<**in**>> (p. 433)* The ID of the member or **DDS_DYNAMIC_DATA_MEMBER_ID_UNSPECIFIED** (p. 105) to look up by name. See **Member Names and IDs** (p. 1022).
value	*<<**in**>> (p. 433)* The value to which to set the member.

Exceptions

One	of the **Standard Return Codes** (p. 532) or **DDS_RETCODE_OUT_OF_RESOURCES** (p. 533)

See also

DDS_DynamicData::get_long (p. 1051)

9.58.6.82 set_short()

```
DDS_ReturnCode_t DDS_DynamicData::set_short (
            const char * member_name,
             DDS_DynamicDataMemberId member_id,
             DDS_Short value )
```

Set the value of the given field, which is of type **DDS_Short** (p. 514).

This API is not supported when the DynamicData object is in CDR format and will fail with **DDS_RETCODE_↩ PRECONDITION_NOT_MET** (p. 532). See **DDS_DynamicData::is_cdr** (p. 1035) for more information.

MT Safety:

UNSAFE.

Parameters

member_name	*<<**in**>> (p. 433)* The name of the member or NULL to look up the member by its ID.
member_id	*<<**in**>> (p. 433)* The ID of the member or **DDS_DYNAMIC_DATA_MEMBER_ID_UNSPECIFIED** (p. 105) to look up by name. See **Member Names and IDs** (p. 1022).
value	*<<**in**>> (p. 433)* The value to which to set the member.

Exceptions

One	of the **Standard Return Codes** (p. 532) or **DDS_RETCODE_OUT_OF_RESOURCES** (p. 533)

See also

DDS_DynamicData::get_short (p. 1052)

9.58.6.83 set_ulong()

```
DDS_ReturnCode_t DDS_DynamicData::set_ulong (
            const char * member_name,
             DDS_DynamicDataMemberId member_id,
             DDS_UnsignedLong value )
```

Set the value of the given field, which is of type **DDS_UnsignedLong** (p. 515).

This API is not supported when the DynamicData object is in CDR format and will fail with **DDS_RETCODE_↩ PRECONDITION_NOT_MET** (p. 532). See **DDS_DynamicData::is_cdr** (p. 1035) for more information.

MT Safety:

UNSAFE.

Parameters

member_name	<<***in***>> *(p. 433)* The name of the member or NULL to look up the member by its ID.
member_id	<<***in***>> *(p. 433)* The ID of the member or **DDS_DYNAMIC_DATA_MEMBER_ID_UNSPECIFIED** (p. 105) to look up by name. See **Member Names and IDs** (p. 1022).
value	<<***in***>> *(p. 433)* The value to which to set the member.

Exceptions

One	of the **Standard Return Codes** (p. 532) or **DDS_RETCODE_OUT_OF_RESOURCES** (p. 533)

See also

DDS_DynamicData::get_ulong (p. 1053)

9.58.6.84 set_ushort()

```
DDS_ReturnCode_t DDS_DynamicData::set_ushort (
            const char * member_name,
             DDS_DynamicDataMemberId member_id,
             DDS_UnsignedShort value )
```

Set the value of the given field, which is of type **DDS_UnsignedShort** (p. 514).

This API is not supported when the DynamicData object is in CDR format and will fail with **DDS_RETCODE_↩ PRECONDITION_NOT_MET** (p. 532). See **DDS_DynamicData::is_cdr** (p. 1035) for more information.

MT Safety:

UNSAFE.

Parameters

member_name	<<***in***>> *(p. 433)* The name of the member or NULL to look up the member by its ID.
member_id	<<***in***>> *(p. 433)* The ID of the member or **DDS_DYNAMIC_DATA_MEMBER_ID_UNSPECIFIED** (p. 105) to look up by name. See **Member Names and IDs** (p. 1022).
value	<<***in***>> *(p. 433)* The value to which to set the member.

Exceptions

One	of the **Standard Return Codes** (p. 532) or **DDS_RETCODE_OUT_OF_RESOURCES** (p. 533)

See also

DDS_DynamicData::get_ushort (p. 1054)

9.58.6.85 set_float()

```
DDS_ReturnCode_t DDS_DynamicData::set_float (
            const char * member_name,
             DDS_DynamicDataMemberId member_id,
             DDS_Float value )
```

Set the value of the given field, which is of type **DDS_Float** (p. 515).

This API is not supported when the DynamicData object is in CDR format and will fail with **DDS_RETCODE_↩ PRECONDITION_NOT_MET** (p. 532). See **DDS_DynamicData::is_cdr** (p. 1035) for more information.

MT Safety:

UNSAFE.

Parameters

member_name	*<<**in**>> (p. 433)* The name of the member or NULL to look up the member by its ID.
member_id	*<<**in**>> (p. 433)* The ID of the member or **DDS_DYNAMIC_DATA_MEMBER_ID_UNSPECIFIED** (p. 105) to look up by name. See **Member Names and IDs** (p. 1022).
value	*<<**in**>> (p. 433)* The value to which to set the member.

Exceptions

One	of the **Standard Return Codes** (p. 532) or **DDS_RETCODE_OUT_OF_RESOURCES** (p. 533)

See also

DDS_DynamicData::get_float (p. 1055)

9.58.6.86 set_double()

```
DDS_ReturnCode_t DDS_DynamicData::set_double (
            const char * member_name,
             DDS_DynamicDataMemberId member_id,
             DDS_Double value )
```

Set the value of the given field, which is of type **DDS_Double** (p. 515).

This API is not supported when the DynamicData object is in CDR format and will fail with **DDS_RETCODE_↩ PRECONDITION_NOT_MET** (p. 532). See **DDS_DynamicData::is_cdr** (p. 1035) for more information.

MT Safety:

UNSAFE.

Parameters

member_name	*<<**in**>> (p. 433)* The name of the member or NULL to look up the member by its ID.
member_id	*<<**in**>> (p. 433)* The ID of the member or **DDS_DYNAMIC_DATA_MEMBER_ID_UNSPECIFIED** (p. 105) to look up by name. See **Member Names and IDs** (p. 1022).
value	*<<**in**>> (p. 433)* The value to which to set the member.

Exceptions

One	of the **Standard Return Codes** (p. 532) or **DDS_RETCODE_OUT_OF_RESOURCES** (p. 533)

See also

DDS_DynamicData::get_double (p. 1056)

9.58.6.87 set_boolean()

```
DDS_ReturnCode_t DDS_DynamicData::set_boolean (
            const char * member_name,
             DDS_DynamicDataMemberId member_id,
             DDS_Boolean value )
```

Set the value of the given field, which is of type **DDS_Boolean** (p. 516).

This API is not supported when the DynamicData object is in CDR format and will fail with **DDS_RETCODE_↩ PRECONDITION_NOT_MET** (p. 532). See **DDS_DynamicData::is_cdr** (p. 1035) for more information.

MT Safety:

UNSAFE.

Parameters

member_name	<<***in***>> *(p. 433)* The name of the member or NULL to look up the member by its ID.
member_id	<<***in***>> *(p. 433)* The ID of the member or **DDS_DYNAMIC_DATA_MEMBER_ID_UNSPECIFIED** (p. 105) to look up by name. See **Member Names and IDs** (p. 1022).
value	<<***in***>> *(p. 433)* The value to which to set the member.

Exceptions

One	of the **Standard Return Codes** (p. 532) or **DDS_RETCODE_OUT_OF_RESOURCES** (p. 533)

See also

DDS_DynamicData::get_boolean (p. 1056)

9.58.6.88 set_char()

```
DDS_ReturnCode_t DDS_DynamicData::set_char (
            const char * member_name,
             DDS_DynamicDataMemberId member_id,
             DDS_Char value )
```

Set the value of the given field, which is of type **DDS_Char** (p. 513).

This API is not supported when the DynamicData object is in CDR format and will fail with **DDS_RETCODE_↩ PRECONDITION_NOT_MET** (p. 532). See **DDS_DynamicData::is_cdr** (p. 1035) for more information.

MT Safety:

UNSAFE.

Parameters

member_name	*<<**in**>> (p. 433)* The name of the member or NULL to look up the member by its ID.
member_id	*<<**in**>> (p. 433)* The ID of the member or **DDS_DYNAMIC_DATA_MEMBER_ID_UNSPECIFIED** (p. 105) to look up by name. See **Member Names and IDs** (p. 1022).
value	*<<**in**>> (p. 433)* The value to which to set the member.

Exceptions

One	of the **Standard Return Codes** (p. 532) or **DDS_RETCODE_OUT_OF_RESOURCES** (p. 533)

See also

DDS_DynamicData::get_char (p. 1057)

9.58.6.89 set_octet()

```
DDS_ReturnCode_t DDS_DynamicData::set_octet (
            const char * member_name,
             DDS_DynamicDataMemberId member_id,
             DDS_Octet value )
```

Set the value of the given field, which is of type **DDS_Octet** (p. 514).

This API is not supported when the DynamicData object is in CDR format and will fail with **DDS_RETCODE_↩ PRECONDITION_NOT_MET** (p. 532). See **DDS_DynamicData::is_cdr** (p. 1035) for more information.

MT Safety:

UNSAFE.

Parameters

member_name	*<<**in**>> (p. 433)* The name of the member or NULL to look up the member by its ID.
member_id	*<<**in**>> (p. 433)* The ID of the member or **DDS_DYNAMIC_DATA_MEMBER_ID_UNSPECIFIED** (p. 105) to look up by name. See **Member Names and IDs** (p. 1022).
value	*<<**in**>> (p. 433)* The value to which to set the member.

Exceptions

One	of the **Standard Return Codes** (p. 532) or **DDS_RETCODE_OUT_OF_RESOURCES** (p. 533)

See also

DDS_DynamicData::get_octet (p. 1058)

9.58.6.90 set_longlong()

```
DDS_ReturnCode_t DDS_DynamicData::set_longlong (
            const char * member_name,
             DDS_DynamicDataMemberId member_id,
             DDS_LongLong value )
```

Set the value of the given field, which is of type **DDS_LongLong** (p. 515).

This API is not supported when the DynamicData object is in CDR format and will fail with **DDS_RETCODE_↩ PRECONDITION_NOT_MET** (p. 532). See **DDS_DynamicData::is_cdr** (p. 1035) for more information.

MT Safety:

UNSAFE.

Parameters

member_name	<<***in***>> *(p. 433)* The name of the member or NULL to look up the member by its ID.
member_id	<<***in***>> *(p. 433)* The ID of the member or **DDS_DYNAMIC_DATA_MEMBER_ID_UNSPECIFIED** (p. 105) to look up by name. See **Member Names and IDs** (p. 1022).
value	<<***in***>> *(p. 433)* The value to which to set the member.

Exceptions

One	of the **Standard Return Codes** (p. 532) or **DDS_RETCODE_OUT_OF_RESOURCES** (p. 533)

See also

DDS_DynamicData::get_longlong (p. 1059)

9.58.6.91 set_ulonglong()

```
DDS_ReturnCode_t DDS_DynamicData::set_ulonglong (
            const char * member_name,
             DDS_DynamicDataMemberId member_id,
             DDS_UnsignedLongLong value )
```

Set the value of the given field, which is of type **DDS_UnsignedLongLong** (p. 515).

This API is not supported when the DynamicData object is in CDR format and will fail with **DDS_RETCODE_↩ PRECONDITION_NOT_MET** (p. 532). See **DDS_DynamicData::is_cdr** (p. 1035) for more information.

MT Safety:

UNSAFE.

Parameters

member_name	*<<**in**>> (p. 433)* The name of the member or NULL to look up the member by its ID.
member_id	*<<**in**>> (p. 433)* The ID of the member or **DDS_DYNAMIC_DATA_MEMBER_ID_UNSPECIFIED** (p. 105) to look up by name. See **Member Names and IDs** (p. 1022).
value	*<<**in**>> (p. 433)* The value to which to set the member.

Exceptions

One	of the **Standard Return Codes** (p. 532) or **DDS_RETCODE_OUT_OF_RESOURCES** (p. 533)

See also

DDS_DynamicData::get_ulonglong (p. 1060)

9.58.6.92 set_longdouble()

```
DDS_ReturnCode_t DDS_DynamicData::set_longdouble (
            const char * member_name,
             DDS_DynamicDataMemberId member_id,
             DDS_LongDouble value )
```

Set the value of the given field, which is of type **DDS_LongDouble** (p. 516).

This API is not supported when the DynamicData object is in CDR format and will fail with **DDS_RETCODE_↩ PRECONDITION_NOT_MET** (p. 532). See **DDS_DynamicData::is_cdr** (p. 1035) for more information.

MT Safety:

UNSAFE.

Parameters

member_name	<<***in***>> *(p. 433)* The name of the member or NULL to look up the member by its ID.
member_id	<<***in***>> *(p. 433)* The ID of the member or **DDS_DYNAMIC_DATA_MEMBER_ID_UNSPECIFIED** (p. 105) to look up by name. See **Member Names and IDs** (p. 1022).
value	<<***in***>> *(p. 433)* The value to which to set the member.

Exceptions

One	of the **Standard Return Codes** (p. 532) or **DDS_RETCODE_OUT_OF_RESOURCES** (p. 533)

See also

DDS_DynamicData::get_longdouble (p. 1060)

9.58.6.93 set_wchar()

```
DDS_ReturnCode_t DDS_DynamicData::set_wchar (
            const char * member_name,
             DDS_DynamicDataMemberId member_id,
             DDS_Wchar value )
```

Set the value of the given field, which is of type **DDS_Wchar** (p. 514).

This API is not supported when the DynamicData object is in CDR format and will fail with **DDS_RETCODE_↩ PRECONDITION_NOT_MET** (p. 532). See **DDS_DynamicData::is_cdr** (p. 1035) for more information.

MT Safety:

UNSAFE.

Parameters

member_name	<<***in***>> *(p. 433)* The name of the member or NULL to look up the member by its ID.
member_id	<<***in***>> *(p. 433)* The ID of the member or **DDS_DYNAMIC_DATA_MEMBER_ID_UNSPECIFIED** (p. 105) to look up by name. See **Member Names and IDs** (p. 1022).
value	<<***in***>> *(p. 433)* The value to which to set the member.

Exceptions

One	of the **Standard Return Codes** (p. 532) or **DDS_RETCODE_OUT_OF_RESOURCES** (p. 533)

See also

DDS_DynamicData::get_char (p. 1057)

9.58.6.94 set_string()

```
DDS_ReturnCode_t DDS_DynamicData::set_string (
            const char * member_name,
             DDS_DynamicDataMemberId member_id,
            const char * value )
```

Set the value of the given field of type char∗.

This API is not supported when the DynamicData object is in CDR format and will fail with **DDS_RETCODE_↩ PRECONDITION_NOT_MET** (p. 532). See **DDS_DynamicData::is_cdr** (p. 1035) for more information.

MT Safety:

UNSAFE.

Parameters

member_name	<<***in***>> *(*p. *433)* The name of the member or NULL to look up the member by its ID.
member_id	<<***in***>> *(*p. *433)* The ID of the member or **DDS_DYNAMIC_DATA_MEMBER_ID_UNSPECIFIED** (p. 105) to look up by name. See **Member Names and IDs** (p. 1022).
value	<<***in***>> *(*p. *433)* The value to which to set the member.

Exceptions

One	of the **Standard Return Codes** (p. 532) or **DDS_RETCODE_OUT_OF_RESOURCES** (p. 533)

See also

DDS_DynamicData::get_string (p. 1062)

9.58.6.95 set_wstring()

```
DDS_ReturnCode_t DDS_DynamicData::set_wstring (
            const char * member_name,
             DDS_DynamicDataMemberId member_id,
            const  DDS_Wchar * value )
```

Set the value of the given field of type **DDS_Wchar** (p. 514)*.

This API is not supported when the DynamicData object is in CDR format and will fail with **DDS_RETCODE_↩ PRECONDITION_NOT_MET** (p. 532). See **DDS_DynamicData::is_cdr** (p. 1035) for more information.

MT Safety:

UNSAFE.

Parameters

member_name	<<***in***>> *(p. 433)* The name of the member or NULL to look up the member by its ID.
member_id	<<***in***>> *(p. 433)* The ID of the member or **DDS_DYNAMIC_DATA_MEMBER_ID_UNSPECIFIED** (p. 105) to look up by name. See **Member Names and IDs** (p. 1022).
value	<<***in***>> *(p. 433)* The value to which to set the member.

Exceptions

One	of the **Standard Return Codes** (p. 532) or **DDS_RETCODE_OUT_OF_RESOURCES** (p. 533)

See also

DDS_DynamicData::get_wstring (p. 1063)

9.58.6.96 set_int8()

```
DDS_ReturnCode_t DDS_DynamicData::set_int8 (
            const char * member_name,
             DDS_DynamicDataMemberId member_id,
             DDS_Int8 value )
```

Set the value of the given field, which is of type **DDS_Int8** (p. 514).

This API is not supported when the DynamicData object is in CDR format and will fail with **DDS_RETCODE_↩ PRECONDITION_NOT_MET** (p. 532). See **DDS_DynamicData::is_cdr** (p. 1035) for more information.

MT Safety:

UNSAFE.

Parameters

member_name	<<***in***>> *(p. 433)* The name of the member or NULL to look up the member by its ID.
member_id	<<***in***>> *(p. 433)* The ID of the member or **DDS_DYNAMIC_DATA_MEMBER_ID_UNSPECIFIED** (p. 105) to look up by name. See **Member Names and IDs** (p. 1022).
value	<<***in***>> *(p. 433)* The value to which to set the member.

Exceptions

One	of the **Standard Return Codes** (p. 532) or **DDS_RETCODE_OUT_OF_RESOURCES** (p. 533)

See also

DDS_DynamicData::get_int8 (p. 1064)

9.58.6.97 set_uint8()

```
DDS_ReturnCode_t DDS_DynamicData::set_uint8 (
            const char * member_name,
             DDS_DynamicDataMemberId member_id,
             DDS_UInt8 value )
```

Set the value of the given field, which is of type **DDS_UInt8** (p. 514).

This API is not supported when the DynamicData object is in CDR format and will fail with **DDS_RETCODE_↩ PRECONDITION_NOT_MET** (p. 532). See **DDS_DynamicData::is_cdr** (p. 1035) for more information.

MT Safety:

UNSAFE.

Parameters

member_name	<<***in***>> *(p. 433)* The name of the member or NULL to look up the member by its ID.
member_id	<<***in***>> *(p. 433)* The ID of the member or **DDS_DYNAMIC_DATA_MEMBER_ID_UNSPECIFIED** (p. 105) to look up by name. See **Member Names and IDs** (p. 1022).
value	<<***in***>> *(p. 433)* The value to which to set the member.

Exceptions

One	of the **Standard Return Codes** (p. 532) or **DDS_RETCODE_OUT_OF_RESOURCES** (p. 533)

See also

DDS_DynamicData::get_uint8 (p. 1065)

9.58.6.98 set_complex_member()

```
DDS_ReturnCode_t DDS_DynamicData::set_complex_member (
            const char * member_name,
             DDS_DynamicDataMemberId member_id,
            const  DDS_DynamicData & value )
```

Copy the state of the given **DDS_DynamicData** (p. 1013) object into a member of this object.

This API is not supported when the DynamicData object is in CDR format and will fail with **DDS_RETCODE_↩ PRECONDITION_NOT_MET** (p. 532). See **DDS_DynamicData::is_cdr** (p. 1035) for more information.

The member may be of type kind **DDS_TK_ARRAY** (p. 89), **DDS_TK_SEQUENCE** (p. 89), **DDS_TK_STRUCT** (p. 89), **DDS_TK_VALUE** (p. 89), or **DDS_TK_UNION** (p. 89). It may be specified by name or by ID.

Example: Copying Data

This method can be used with **DDS_DynamicData::bind_complex_member** (p. 1042) to copy from one **DDS_↩ DynamicData** (p. 1013) object to another efficiently. Suppose the following data structure:

```
struct Bar {

    short theShort;

};


struct Foo {

    Bar theBar;

};
```

Suppose we have two instances of **Foo** (p. 1916), foo_dst and foo_src, and we want to replace the contents of foo_dst.theBar with the contents of foo_src.theBar. The following example shows how to do this (error handling has been omitted for the sake of brevity).

```
DDS_DynamicData* foo_dst = ...;

DDS_DynamicData* foo_src = ...;

DDS_DynamicData* bar = new DDS_DynamicData(NULL, myProperties);

// Point to the source of the copy:
foo_src->bind_complex_member(
    bar,
    "theBar",
    DDS_DYNAMIC_DATA_MEMBER_ID_UNSPECIFIED);

// Just one copy:
foo_dst->set_complex_member(
    "theBar",
    DDS_DYNAMIC_DATA_MEMBER_ID_UNSPECIFIED,
    bar);
// Tear down:

foo_src->unbind_complex_member(bar);

delete bar;
```

MT Safety:

UNSAFE.

Parameters

member_name	<<***in***>> *(p. 433)* The name of the member or NULL to look up the member by its ID.
member_id	<<***in***>> *(p. 433)* The ID of the member or **DDS_DYNAMIC_DATA_MEMBER_ID_UNSPECIFIED** (p. 105) to look up by name. See **Member Names and IDs** (p. 1022).
value	<<***in***>> *(p. 433)* The source **DDS_DynamicData** (p. 1013) object whose contents will be copied.

Exceptions

One	of the **Standard Return Codes** (p. 532) or **DDS_RETCODE_OUT_OF_RESOURCES** (p. 533)

See also

DDS_DynamicData::get_complex_member (p. 1066)

DDS_DynamicData::bind_complex_member (p. 1042)

9.58.6.99 set_long_array()

```
DDS_ReturnCode_t DDS_DynamicData::set_long_array (
            const char * member_name,
             DDS_DynamicDataMemberId member_id,
             DDS_UnsignedLong length,
            const  DDS_Long * array )
```

Set the contents of the given array member. The array may contain members of type **DDS_Long** (p. 515) or **DDS_Enum** (p. 516).

This method will perform an automatic conversion to **DDS_LongSeq** (p. 1192).

If the destination array is insufficiently long to store the data, this operation will fail without copying anything.

This API is not supported when the DynamicData object is in CDR format and will fail with **DDS_RETCODE_↩ PRECONDITION_NOT_MET** (p. 532). See **DDS_DynamicData::is_cdr** (p. 1035) for more information.

MT Safety:

UNSAFE.

Parameters

member_name	<<***in***>> *(p. 433)* The name of the member or NULL to look up the member by its ID.
member_id	<<***in***>> *(p. 433)* The ID of the member or **DDS_DYNAMIC_DATA_MEMBER_ID_UNSPECIFIED** (p. 105) to look up by name. See **Member Names and IDs** (p. 1022).
length	<<***in***>> *(p. 433)* The length of `array`.
array	<<***in***>> *(p. 433)* The elements to copy.

Exceptions

One	of the **Standard Return Codes** (p. 532) or **DDS_RETCODE_OUT_OF_RESOURCES** (p. 533)

See also

DDS_DynamicData::get_long_array (p. 1067)

DDS_DynamicData::set_long_seq (p. 1120)

9.58.6.100 set_short_array()

```
DDS_ReturnCode_t DDS_DynamicData::set_short_array (
            const char * member_name,
             DDS_DynamicDataMemberId member_id,
             DDS_UnsignedLong length,
            const  DDS_Short * array )
```

Set the contents of the given array member.

This method will perform an automatic conversion to **DDS_ShortSeq** (p. 1363).

If the destination array is insufficiently long to store the data, this operation will fail without copying anything.

This API is not supported when the DynamicData object is in CDR format and will fail with **DDS_RETCODE_↩ PRECONDITION_NOT_MET** (p. 532). See **DDS_DynamicData::is_cdr** (p. 1035) for more information.

MT Safety:

UNSAFE.

Parameters

member_name	<<***in***>> *(p. 433)* The name of the member or NULL to look up the member by its ID.
member_id	<<***in***>> *(p. 433)* The ID of the member or **DDS_DYNAMIC_DATA_MEMBER_ID_UNSPECIFIED** (p. 105) to look up by name. See **Member Names and IDs** (p. 1022).
length	<<***in***>> *(p. 433)* The length of `array`.
array	<<***in***>> *(p. 433)* The elements to copy.

Exceptions

One	of the **Standard Return Codes** (p. 532) or **DDS_RETCODE_OUT_OF_RESOURCES** (p. 533)

See also

DDS_DynamicData::get_short_array (p. 1068)

DDS_DynamicData::set_short_seq (p. 1121)

9.58.6.101 set_ulong_array()

```
DDS_ReturnCode_t DDS_DynamicData::set_ulong_array (
            const char * member_name,
             DDS_DynamicDataMemberId member_id,
             DDS_UnsignedLong length,
            const  DDS_UnsignedLong * array )
```

Set the contents of the given array member. The array may contain members of type **DDS_UnsignedLong** (p. 515) or **DDS_Enum** (p. 516).

This method will perform an automatic conversion to **DDS_UnsignedLongSeq** (p. 1503).

If the destination array is insufficiently long to store the data, this operation will fail without copying anything.

This API is not supported when the DynamicData object is in CDR format and will fail with **DDS_RETCODE_↩ PRECONDITION_NOT_MET** (p. 532). See **DDS_DynamicData::is_cdr** (p. 1035) for more information.

MT Safety:

UNSAFE.

Parameters

member_name	<<***in***>> *(p. 433)* The name of the member or NULL to look up the member by its ID.
member_id	<<***in***>> *(p. 433)* The ID of the member or **DDS_DYNAMIC_DATA_MEMBER_ID_UNSPECIFIED** (p. 105) to look up by name. See **Member Names and IDs** (p. 1022).
length	<<***in***>> *(p. 433)* The length of `array`.
array	<<***in***>> *(p. 433)* The elements to copy.

Exceptions

One	of the **Standard Return Codes** (p. 532) or **DDS_RETCODE_OUT_OF_RESOURCES** (p. 533)

See also

DDS_DynamicData::get_ulong_array (p. 1069)

DDS_DynamicData::set_ulong_seq (p. 1121)

9.58.6.102 set_ushort_array()

```
DDS_ReturnCode_t DDS_DynamicData::set_ushort_array (
            const char * member_name,
             DDS_DynamicDataMemberId member_id,
             DDS_UnsignedLong length,
            const  DDS_UnsignedShort * array )
```

Set the contents of the given array member.

This method will perform an automatic conversion to **DDS_UnsignedShortSeq** (p. 1503).

If the destination array is insufficiently long to store the data, this operation will fail without copying anything.

This API is not supported when the DynamicData object is in CDR format and will fail with **DDS_RETCODE_↩ PRECONDITION_NOT_MET** (p. 532). See **DDS_DynamicData::is_cdr** (p. 1035) for more information.

MT Safety:

UNSAFE.

Parameters

member_name	<<***in***>> *(*p. *433)* The name of the member or NULL to look up the member by its ID.
member_id	<<***in***>> *(*p. *433)* The ID of the member or **DDS_DYNAMIC_DATA_MEMBER_ID_UNSPECIFIED** (p. 105) to look up by name. See **Member Names and IDs** (p. 1022).
length	<<***in***>> *(*p. *433)* The length of `array`.
array	<<***in***>> *(*p. *433)* The elements to copy.

Exceptions

One	of the **Standard Return Codes** (p. 532) or **DDS_RETCODE_OUT_OF_RESOURCES** (p. 533)

See also

DDS_DynamicData::get_ushort_array (p. 1070)

DDS_DynamicData::set_ushort_seq (p. 1122)

9.58.6.103 set_float_array()

```
DDS_ReturnCode_t DDS_DynamicData::set_float_array (
            const char * member_name,
             DDS_DynamicDataMemberId member_id,
```

```
 DDS_UnsignedLong length,
const  DDS_Float * array )
```

Set the contents of the given array member.

This method will perform an automatic conversion to **DDS_FloatSeq** (p. 1156).

If the destination array is insufficiently long to store the data, this operation will fail without copying anything.

This API is not supported when the DynamicData object is in CDR format and will fail with **DDS_RETCODE_↩ PRECONDITION_NOT_MET** (p. 532). See **DDS_DynamicData::is_cdr** (p. 1035) for more information.

MT Safety:

UNSAFE.

Parameters

member_name	<<***in***>> *(p. 433)* The name of the member or NULL to look up the member by its ID.
member_id	<<***in***>> *(p. 433)* The ID of the member or **DDS_DYNAMIC_DATA_MEMBER_ID_UNSPECIFIED** (p. 105) to look up by name. See **Member Names and IDs** (p. 1022).
length	<<***in***>> *(p. 433)* The length of `array`.
array	<<***in***>> *(p. 433)* The elements to copy.

Exceptions

One	of the **Standard Return Codes** (p. 532) or **DDS_RETCODE_OUT_OF_RESOURCES** (p. 533)

See also

DDS_DynamicData::get_float_array (p. 1071)

DDS_DynamicData::set_float_seq (p. 1123)

9.58.6.104 set_double_array()

```
DDS_ReturnCode_t DDS_DynamicData::set_double_array (
            const char * member_name,
             DDS_DynamicDataMemberId member_id,
             DDS_UnsignedLong length,
            const  DDS_Double * array )
```

Set the contents of the given array member.

This method will perform an automatic conversion to **DDS_DoubleSeq** (p. 1004).

If the destination array is insufficiently long to store the data, this operation will fail without copying anything.

This API is not supported when the DynamicData object is in CDR format and will fail with **DDS_RETCODE_↩ PRECONDITION_NOT_MET** (p. 532). See **DDS_DynamicData::is_cdr** (p. 1035) for more information.

MT Safety:

UNSAFE.

Parameters

member_name	<<***in***>> *(p. 433)* The name of the member or NULL to look up the member by its ID.
member_id	<<***in***>> *(p. 433)* The ID of the member or **DDS_DYNAMIC_DATA_MEMBER_ID_UNSPECIFIED** (p. 105) to look up by name. See **Member Names and IDs** (p. 1022).
length	<<***in***>> *(p. 433)* The length of `array`.
array	<<***in***>> *(p. 433)* The elements to copy.

Exceptions

One	of the **Standard Return Codes** (p. 532) or **DDS_RETCODE_OUT_OF_RESOURCES** (p. 533)

See also

DDS_DynamicData::get_double_array (p. 1072)

DDS_DynamicData::set_double_seq (p. 1124)

9.58.6.105 set_boolean_array()

```
DDS_ReturnCode_t DDS_DynamicData::set_boolean_array (
            const char * member_name,
             DDS_DynamicDataMemberId member_id,
             DDS_UnsignedLong length,
            const  DDS_Boolean * array )
```

Set the contents of the given array member.

This method will perform an automatic conversion to **DDS_BooleanSeq** (p. 814).

If the destination array is insufficiently long to store the data, this operation will fail without copying anything.

This API is not supported when the DynamicData object is in CDR format and will fail with **DDS_RETCODE_↩ PRECONDITION_NOT_MET** (p. 532). See **DDS_DynamicData::is_cdr** (p. 1035) for more information.

MT Safety:

UNSAFE.

Parameters

member_name	<<***in***>> *(p. 433)* The name of the member or NULL to look up the member by its ID.
member_id	<<***in***>> *(p. 433)* The ID of the member or **DDS_DYNAMIC_DATA_MEMBER_ID_UNSPECIFIED** (p. 105) to look up by name. See **Member Names and IDs** (p. 1022).
length	<<***in***>> *(p. 433)* The length of `array`.
array	<<***in***>> *(p. 433)* The elements to copy.

Exceptions

One	of the **Standard Return Codes** (p. 532) or **DDS_RETCODE_OUT_OF_RESOURCES** (p. 533)

See also

DDS_DynamicData::get_boolean_array (p. 1073)

DDS_DynamicData::set_boolean_seq (p. 1125)

9.58.6.106 set_char_array()

```
DDS_ReturnCode_t DDS_DynamicData::set_char_array (
            const char * member_name,
             DDS_DynamicDataMemberId member_id,
             DDS_UnsignedLong length,
            const  DDS_Char * array )
```

Set the contents of the given array member.

This method will perform an automatic conversion to **DDS_CharSeq** (p. 824).

If the destination array is insufficiently long to store the data, this operation will fail without copying anything.

This API is not supported when the DynamicData object is in CDR format and will fail with **DDS_RETCODE_↩ PRECONDITION_NOT_MET** (p. 532). See **DDS_DynamicData::is_cdr** (p. 1035) for more information.

MT Safety:

UNSAFE.

Parameters

member_name	<<***in***>> *(p. 433)* The name of the member or NULL to look up the member by its ID.
member_id	<<***in***>> *(p. 433)* The ID of the member or **DDS_DYNAMIC_DATA_MEMBER_ID_UNSPECIFIED** (p. 105) to look up by name. See **Member Names and IDs** (p. 1022).
length	<<***in***>> *(p. 433)* The length of `array`.
array	<<***in***>> *(p. 433)* The elements to copy.

Exceptions

One	of the **Standard Return Codes** (p. 532) or **DDS_RETCODE_OUT_OF_RESOURCES** (p. 533)

See also

DDS_DynamicData::get_char_array (p. 1074)

DDS_DynamicData::set_char_seq (p. 1125)

9.58.6.107 set_octet_array()

```
DDS_ReturnCode_t DDS_DynamicData::set_octet_array (
            const char * member_name,
             DDS_DynamicDataMemberId member_id,
             DDS_UnsignedLong length,
            const  DDS_Octet * array )
```

Set the contents of the given array member.

This method will perform an automatic conversion to **DDS_OctetSeq** (p. 1218).

If the destination array is insufficiently long to store the data, this operation will fail without copying anything.

This API is not supported when the DynamicData object is in CDR format and will fail with **DDS_RETCODE_↩ PRECONDITION_NOT_MET** (p. 532). See **DDS_DynamicData::is_cdr** (p. 1035) for more information.

MT Safety:

UNSAFE.

Parameters

member_name	<<***in***>> *(p. 433)* The name of the member or NULL to look up the member by its ID.
member_id	<<***in***>> *(p. 433)* The ID of the member or **DDS_DYNAMIC_DATA_MEMBER_ID_UNSPECIFIED** (p. 105) to look up by name. See **Member Names and IDs** (p. 1022).
length	<<***in***>> *(p. 433)* The length of `array`.
array	<<***in***>> *(p. 433)* The elements to copy.

Exceptions

One	of the **Standard Return Codes** (p. 532) or **DDS_RETCODE_OUT_OF_RESOURCES** (p. 533)

See also

DDS_DynamicData::get_octet_array (p. 1075)

DDS_DynamicData::set_octet_seq (p. 1126)

9.58.6.108 set_longlong_array()

```
DDS_ReturnCode_t DDS_DynamicData::set_longlong_array (
            const char * member_name,
             DDS_DynamicDataMemberId member_id,
             DDS_UnsignedLong length,
            const  DDS_LongLong * array )
```

Set the contents of the given array member.

This method will perform an automatic conversion to **DDS_LongLongSeq** (p. 1192).

If the destination array is insufficiently long to store the data, this operation will fail without copying anything.

This API is not supported when the DynamicData object is in CDR format and will fail with **DDS_RETCODE_↩ PRECONDITION_NOT_MET** (p. 532). See **DDS_DynamicData::is_cdr** (p. 1035) for more information.

MT Safety:

UNSAFE.

Parameters

member_name	<<***in***>> *(p. 433)* The name of the member or NULL to look up the member by its ID.
member_id	<<***in***>> *(p. 433)* The ID of the member or **DDS_DYNAMIC_DATA_MEMBER_ID_UNSPECIFIED** (p. 105) to look up by name. See **Member Names and IDs** (p. 1022).
length	<<***in***>> *(p. 433)* The length of `array`.
array	<<***in***>> *(p. 433)* The elements to copy.

Exceptions

One	of the **Standard Return Codes** (p. 532) or **DDS_RETCODE_OUT_OF_RESOURCES** (p. 533)

See also

DDS_DynamicData::get_longlong_array (p. 1075)

DDS_DynamicData::set_longlong_seq (p. 1127)

9.58.6.109 set_ulonglong_array()

```
DDS_ReturnCode_t DDS_DynamicData::set_ulonglong_array (
            const char * member_name,
             DDS_DynamicDataMemberId member_id,
             DDS_UnsignedLong length,
            const  DDS_UnsignedLongLong * array )
```

Set the contents of the given array member.

This method will perform an automatic conversion to **DDS_UnsignedLongLongSeq** (p. 1503).

If the destination array is insufficiently long to store the data, this operation will fail without copying anything.

This API is not supported when the DynamicData object is in CDR format and will fail with **DDS_RETCODE_↩ PRECONDITION_NOT_MET** (p. 532). See **DDS_DynamicData::is_cdr** (p. 1035) for more information.

MT Safety:

UNSAFE.

Parameters

member_name	<<***in***>> *(*p. *433)* The name of the member or NULL to look up the member by its ID.
member_id	<<***in***>> *(*p. *433)* The ID of the member or **DDS_DYNAMIC_DATA_MEMBER_ID_UNSPECIFIED** (p. 105) to look up by name. See **Member Names and IDs** (p. 1022).
length	<<***in***>> *(*p. *433)* The length of `array`.
array	<<***in***>> *(*p. *433)* The elements to copy.

Exceptions

One	of the **Standard Return Codes** (p. 532) or **DDS_RETCODE_OUT_OF_RESOURCES** (p. 533)

See also

DDS_DynamicData::get_ulonglong_array (p. 1076)

DDS_DynamicData::set_ulonglong_seq (p. 1128)

9.58.6.110 set_longdouble_array()

```
DDS_ReturnCode_t DDS_DynamicData::set_longdouble_array (
            const char * member_name,
             DDS_DynamicDataMemberId member_id,
```

```
 DDS_UnsignedLong length,
const  DDS_LongDouble * array )
```

Set the contents of the given array member.

This method will perform an automatic conversion to **DDS_LongDoubleSeq** (p. 1192).

If the destination array is insufficiently long to store the data, this operation will fail without copying anything.

This API is not supported when the DynamicData object is in CDR format and will fail with **DDS_RETCODE_↩ PRECONDITION_NOT_MET** (p. 532). See **DDS_DynamicData::is_cdr** (p. 1035) for more information.

MT Safety:

UNSAFE.

Parameters

member_name	<<***in***>> *(p. 433)* The name of the member or NULL to look up the member by its ID.
member_id	<<***in***>> *(p. 433)* The ID of the member or **DDS_DYNAMIC_DATA_MEMBER_ID_UNSPECIFIED** (p. 105) to look up by name. See **Member Names and IDs** (p. 1022).
length	<<***in***>> *(p. 433)* The length of `array`.
array	<<***in***>> *(p. 433)* The elements to copy.

Exceptions

One	of the **Standard Return Codes** (p. 532) or **DDS_RETCODE_OUT_OF_RESOURCES** (p. 533)

See also

DDS_DynamicData::get_longdouble_array (p. 1077)

DDS_DynamicData::set_longdouble_seq (p. 1129)

9.58.6.111 set_wchar_array()

```
DDS_ReturnCode_t DDS_DynamicData::set_wchar_array (
            const char * member_name,
             DDS_DynamicDataMemberId member_id,
             DDS_UnsignedLong length,
            const  DDS_Wchar * array )
```

Set the contents of the given array member.

This method will perform an automatic conversion to **DDS_WcharSeq** (p. 1510).

If the destination array is insufficiently long to store the data, this operation will fail without copying anything.

This API is not supported when the DynamicData object is in CDR format and will fail with **DDS_RETCODE_↩ PRECONDITION_NOT_MET** (p. 532). See **DDS_DynamicData::is_cdr** (p. 1035) for more information.

MT Safety:

UNSAFE.

Parameters

member_name	<<***in***>> *(p. 433)* The name of the member or NULL to look up the member by its ID.
member_id	<<***in***>> *(p. 433)* The ID of the member or **DDS_DYNAMIC_DATA_MEMBER_ID_UNSPECIFIED** (p. 105) to look up by name. See **Member Names and IDs** (p. 1022).
length	<<***in***>> *(p. 433)* The length of `array`.
array	<<***in***>> *(p. 433)* The elements to copy.

Exceptions

One	of the **Standard Return Codes** (p. 532) or **DDS_RETCODE_OUT_OF_RESOURCES** (p. 533)

See also

DDS_DynamicData::get_wchar_array (p. 1078)

DDS_DynamicData::set_wchar_seq (p. 1129)

9.58.6.112 set_int8_array()

```
DDS_ReturnCode_t DDS_DynamicData::set_int8_array (
            const char * member_name,
             DDS_DynamicDataMemberId member_id,
             DDS_UnsignedLong length,
            const  DDS_Int8 * array )
```

Set the contents of the given array member.

This method will perform an automatic conversion to **DDS_Int8Seq** (p. 1167).

If the destination array is insufficiently long to store the data, this operation will fail without copying anything.

This API is not supported when the DynamicData object is in CDR format and will fail with **DDS_RETCODE_↩ PRECONDITION_NOT_MET** (p. 532). See **DDS_DynamicData::is_cdr** (p. 1035) for more information.

MT Safety:

UNSAFE.

Parameters

member_name	<<***in***>> *(p. 433)* The name of the member or NULL to look up the member by its ID.
member_id	<<***in***>> *(p. 433)* The ID of the member or **DDS_DYNAMIC_DATA_MEMBER_ID_UNSPECIFIED** (p. 105) to look up by name. See **Member Names and IDs** (p. 1022).
length	<<***in***>> *(p. 433)* The length of `array`.
array	<<***in***>> *(p. 433)* The elements to copy.

Exceptions

One	of the **Standard Return Codes** (p. 532) or **DDS_RETCODE_OUT_OF_RESOURCES** (p. 533)

See also

DDS_DynamicData::get_int8_array (p. 1079)

DDS_DynamicData::set_int8_seq (p. 1130)

9.58.6.113 set_uint8_array()

```
DDS_ReturnCode_t DDS_DynamicData::set_uint8_array (
            const char * member_name,
             DDS_DynamicDataMemberId member_id,
             DDS_UnsignedLong length,
            const  DDS_UInt8 * array )
```

Set the contents of the given array member.

This method will perform an automatic conversion to **DDS_UInt8Seq** (p. 1500).

If the destination array is insufficiently long to store the data, this operation will fail without copying anything.

This API is not supported when the DynamicData object is in CDR format and will fail with **DDS_RETCODE_↩ PRECONDITION_NOT_MET** (p. 532). See **DDS_DynamicData::is_cdr** (p. 1035) for more information.

MT Safety:

UNSAFE.

Parameters

member_name	<<***in***>> *(p. 433)* The name of the member or NULL to look up the member by its ID.
member_id	<<***in***>> *(p. 433)* The ID of the member or **DDS_DYNAMIC_DATA_MEMBER_ID_UNSPECIFIED** (p. 105) to look up by name. See **Member Names and IDs** (p. 1022).
length	<<***in***>> *(p. 433)* The length of `array`.
array	<<***in***>> *(p. 433)* The elements to copy.

Exceptions

One	of the **Standard Return Codes** (p. 532) or **DDS_RETCODE_OUT_OF_RESOURCES** (p. 533)

See also

DDS_DynamicData::get_uint8_array (p. 1080)

DDS_DynamicData::set_uint8_seq (p. 1131)

9.58.6.114 set_long_seq()

```
DDS_ReturnCode_t DDS_DynamicData::set_long_seq (
            const char * member_name,
             DDS_DynamicDataMemberId member_id,
            const  DDS_LongSeq & value )
```

Set the contents of the given sequence member.

This method will perform an automatic conversion to an array of **DDS_Long** (p. 515).

This API is not supported when the DynamicData object is in CDR format and will fail with **DDS_RETCODE_↩ PRECONDITION_NOT_MET** (p. 532). See **DDS_DynamicData::is_cdr** (p. 1035) for more information.

MT Safety:

UNSAFE.

Parameters

member_name	*<<**in**>> (p. 433)* The name of the member or NULL to look up the member by its ID.
member_id	*<<**in**>> (p. 433)* The ID of the member or **DDS_DYNAMIC_DATA_MEMBER_ID_UNSPECIFIED** (p. 105) to look up by name. See **Member Names and IDs** (p. 1022).
value	*<<**in**>> (p. 433)* A sequence, from which the elements will be copied.

Exceptions

One	of the **Standard Return Codes** (p. 532) or **DDS_RETCODE_OUT_OF_RESOURCES** (p. 533)

See also

DDS_DynamicData::get_long_seq (p. 1081)

DDS_DynamicData::set_long_array (p. 1107)

9.58.6.115 set_short_seq()

```
DDS_ReturnCode_t DDS_DynamicData::set_short_seq (
            const char * member_name,
             DDS_DynamicDataMemberId member_id,
            const  DDS_ShortSeq & value )
```

Set the contents of the given sequence member.

This method will perform an automatic conversion to an array of **DDS_Short** (p. 514).

This API is not supported when the DynamicData object is in CDR format and will fail with **DDS_RETCODE_↩ PRECONDITION_NOT_MET** (p. 532). See **DDS_DynamicData::is_cdr** (p. 1035) for more information.

MT Safety:

UNSAFE.

Parameters

member_name	<<***in***>> *(p. 433)* The name of the member or NULL to look up the member by its ID.
member_id	<<***in***>> *(p. 433)* The ID of the member or **DDS_DYNAMIC_DATA_MEMBER_ID_UNSPECIFIED** (p. 105) to look up by name. See **Member Names and IDs** (p. 1022).
value	<<***in***>> *(p. 433)* A sequence, from which the elements will be copied.

Exceptions

One	of the **Standard Return Codes** (p. 532) or **DDS_RETCODE_OUT_OF_RESOURCES** (p. 533)

See also

DDS_DynamicData::get_short_seq (p. 1082)

DDS_DynamicData::set_short_array (p. 1108)

9.58.6.116 set_ulong_seq()

```
DDS_ReturnCode_t DDS_DynamicData::set_ulong_seq (
            const char * member_name,
             DDS_DynamicDataMemberId member_id,
            const  DDS_UnsignedLongSeq & value )
```

Set the contents of the given sequence member.

This method will perform an automatic conversion to an array of **DDS_UnsignedLong** (p. 515).

This API is not supported when the DynamicData object is in CDR format and will fail with **DDS_RETCODE_↩ PRECONDITION_NOT_MET** (p. 532). See **DDS_DynamicData::is_cdr** (p. 1035) for more information.

MT Safety:

UNSAFE.

Parameters

member_name	<<***in***>> *(p. 433)* The name of the member or NULL to look up the member by its ID.
member_id	<<***in***>> *(p. 433)* The ID of the member or **DDS_DYNAMIC_DATA_MEMBER_ID_UNSPECIFIED** (p. 105) to look up by name. See **Member Names and IDs** (p. 1022).
value	<<***in***>> *(p. 433)* A sequence, from which the elements will be copied.

Exceptions

One	of the **Standard Return Codes** (p. 532) or **DDS_RETCODE_OUT_OF_RESOURCES** (p. 533)

See also

DDS_DynamicData::get_ulong_seq (p. 1082)

DDS_DynamicData::set_ulong_array (p. 1109)

9.58.6.117 set_ushort_seq()

```
DDS_ReturnCode_t DDS_DynamicData::set_ushort_seq (
            const char * member_name,
             DDS_DynamicDataMemberId member_id,
            const  DDS_UnsignedShortSeq & value )
```

Set the contents of the given sequence member.

This method will perform an automatic conversion to an array of **DDS_UnsignedShort** (p. 514).

This API is not supported when the DynamicData object is in CDR format and will fail with **DDS_RETCODE_↩ PRECONDITION_NOT_MET** (p. 532). See **DDS_DynamicData::is_cdr** (p. 1035) for more information.

MT Safety:

UNSAFE.

Parameters

member_name	<<***in***>> *(p. 433)* The name of the member or NULL to look up the member by its ID.
member_id	<<***in***>> *(p. 433)* The ID of the member or **DDS_DYNAMIC_DATA_MEMBER_ID_UNSPECIFIED** (p. 105) to look up by name. See **Member Names and IDs** (p. 1022).
value	<<***in***>> *(p. 433)* A sequence, from which the elements will be copied.

Exceptions

One	of the **Standard Return Codes** (p. 532) or **DDS_RETCODE_OUT_OF_RESOURCES** (p. 533)

See also

DDS_DynamicData::get_ushort_seq (p. 1083)

DDS_DynamicData::set_ushort_array (p. 1109)

9.58.6.118 set_float_seq()

```
DDS_ReturnCode_t DDS_DynamicData::set_float_seq (
            const char * member_name,
             DDS_DynamicDataMemberId member_id,
            const  DDS_FloatSeq & value )
```

Set the contents of the given sequence member.

This method will perform an automatic conversion to an array of **DDS_Float** (p. 515).

This API is not supported when the DynamicData object is in CDR format and will fail with **DDS_RETCODE_↩ PRECONDITION_NOT_MET** (p. 532). See **DDS_DynamicData::is_cdr** (p. 1035) for more information.

MT Safety:

UNSAFE.

Parameters

member_name	<<***in***>> *(p. 433)* The name of the member or NULL to look up the member by its ID.
member_id	<<***in***>> *(p. 433)* The ID of the member or **DDS_DYNAMIC_DATA_MEMBER_ID_UNSPECIFIED** (p. 105) to look up by name. See **Member Names and IDs** (p. 1022).
value	<<***in***>> *(p. 433)* A sequence, from which the elements will be copied.

Exceptions

One	of the **Standard Return Codes** (p. 532) or **DDS_RETCODE_OUT_OF_RESOURCES** (p. 533)

See also

DDS_DynamicData::get_float_seq (p. 1084)

DDS_DynamicData::set_float_array (p. 1110)

9.58.6.119 set_double_seq()

```
DDS_ReturnCode_t DDS_DynamicData::set_double_seq (
            const char * member_name,
             DDS_DynamicDataMemberId member_id,
            const  DDS_DoubleSeq & value )
```

Set the contents of the given sequence member.

This method will perform an automatic conversion to an array of **DDS_Double** (p. 515).

This API is not supported when the DynamicData object is in CDR format and will fail with **DDS_RETCODE_↩ PRECONDITION_NOT_MET** (p. 532). See **DDS_DynamicData::is_cdr** (p. 1035) for more information.

MT Safety:

UNSAFE.

Parameters

member_name	<<***in***>> *(p. 433)* The name of the member or NULL to look up the member by its ID.
member_id	<<***in***>> *(p. 433)* The ID of the member or **DDS_DYNAMIC_DATA_MEMBER_ID_UNSPECIFIED** (p. 105) to look up by name. See **Member Names and IDs** (p. 1022).
value	<<***in***>> *(p. 433)* A sequence, from which the elements will be copied.

Exceptions

One	of the **Standard Return Codes** (p. 532) or **DDS_RETCODE_OUT_OF_RESOURCES** (p. 533)

See also

DDS_DynamicData::get_double_seq (p. 1085)

DDS_DynamicData::set_double_array (p. 1111)

9.58.6.120 set_boolean_seq()

```
DDS_ReturnCode_t DDS_DynamicData::set_boolean_seq (
            const char * member_name,
             DDS_DynamicDataMemberId member_id,
            const  DDS_BooleanSeq & value )
```

Set the contents of the given sequence member.

This method will perform an automatic conversion to an array of **DDS_Boolean** (p. 516).

This API is not supported when the DynamicData object is in CDR format and will fail with **DDS_RETCODE_↩ PRECONDITION_NOT_MET** (p. 532). See **DDS_DynamicData::is_cdr** (p. 1035) for more information.

MT Safety:

UNSAFE.

Parameters

member_name	<<***in***>> *(p. 433)* The name of the member or NULL to look up the member by its ID.
member_id	<<***in***>> *(p. 433)* The ID of the member or **DDS_DYNAMIC_DATA_MEMBER_ID_UNSPECIFIED** (p. 105) to look up by name. See **Member Names and IDs** (p. 1022).
value	<<***in***>> *(p. 433)* A sequence, from which the elements will be copied.

Exceptions

One	of the **Standard Return Codes** (p. 532) or **DDS_RETCODE_OUT_OF_RESOURCES** (p. 533)

See also

DDS_DynamicData::get_boolean_seq (p. 1086)

DDS_DynamicData::set_boolean_array (p. 1112)

9.58.6.121 set_char_seq()

```
DDS_ReturnCode_t DDS_DynamicData::set_char_seq (
            const char * member_name,
             DDS_DynamicDataMemberId member_id,
            const  DDS_CharSeq & value )
```

Set the contents of the given sequence member.

This method will perform an automatic conversion to an array of **DDS_Char** (p. 513).

This API is not supported when the DynamicData object is in CDR format and will fail with **DDS_RETCODE_↩ PRECONDITION_NOT_MET** (p. 532). See **DDS_DynamicData::is_cdr** (p. 1035) for more information.

MT Safety:

UNSAFE.

Parameters

member_name	<<***in***>> *(p. 433)* The name of the member or NULL to look up the member by its ID.
member_id	<<***in***>> *(p. 433)* The ID of the member or **DDS_DYNAMIC_DATA_MEMBER_ID_UNSPECIFIED** (p. 105) to look up by name. See **Member Names and IDs** (p. 1022).
value	<<***in***>> *(p. 433)* A sequence, from which the elements will be copied.

Exceptions

One	of the **Standard Return Codes** (p. 532) or **DDS_RETCODE_OUT_OF_RESOURCES** (p. 533)

See also

DDS_DynamicData::get_char_seq (p. 1087)

DDS_DynamicData::set_char_array (p. 1113)

9.58.6.122 set_octet_seq()

```
DDS_ReturnCode_t DDS_DynamicData::set_octet_seq (
            const char * member_name,
             DDS_DynamicDataMemberId member_id,
            const  DDS_OctetSeq & value )
```

Set the contents of the given sequence member.

This method will perform an automatic conversion to an array of **DDS_Octet** (p. 514).

This API is not supported when the DynamicData object is in CDR format and will fail with **DDS_RETCODE_↩ PRECONDITION_NOT_MET** (p. 532). See **DDS_DynamicData::is_cdr** (p. 1035) for more information.

MT Safety:

UNSAFE.

Parameters

member_name	<<***in***>> *(p. 433)* The name of the member or NULL to look up the member by its ID.
member_id	<<***in***>> *(p. 433)* The ID of the member or **DDS_DYNAMIC_DATA_MEMBER_ID_UNSPECIFIED** (p. 105) to look up by name. See **Member Names and IDs** (p. 1022).
value	<<***in***>> *(p. 433)* A sequence, from which the elements will be copied.

Exceptions

One	of the **Standard Return Codes** (p. 532) or **DDS_RETCODE_OUT_OF_RESOURCES** (p. 533)

See also

DDS_DynamicData::get_octet_seq (p. 1087)

DDS_DynamicData::set_octet_array (p. 1114)

9.58.6.123 set_longlong_seq()

```
DDS_ReturnCode_t DDS_DynamicData::set_longlong_seq (
            const char * member_name,
             DDS_DynamicDataMemberId member_id,
            const  DDS_LongLongSeq & value )
```

Set the contents of the given sequence member.

This method will perform an automatic conversion to an array of **DDS_LongLong** (p. 515).

This API is not supported when the DynamicData object is in CDR format and will fail with **DDS_RETCODE_↩ PRECONDITION_NOT_MET** (p. 532). See **DDS_DynamicData::is_cdr** (p. 1035) for more information.

MT Safety:

UNSAFE.

Parameters

member_name	<<***in***>> *(p. 433)* The name of the member or NULL to look up the member by its ID.
member_id	<<***in***>> *(p. 433)* The ID of the member or **DDS_DYNAMIC_DATA_MEMBER_ID_UNSPECIFIED** (p. 105) to look up by name. See **Member Names and IDs** (p. 1022).
value	<<***in***>> *(p. 433)* A sequence, from which the elements will be copied.

Exceptions

One	of the **Standard Return Codes** (p. 532) or **DDS_RETCODE_OUT_OF_RESOURCES** (p. 533)

See also

DDS_DynamicData::get_longlong_seq (p. 1088)

DDS_DynamicData::set_longlong_array (p. 1115)

9.58.6.124 set_ulonglong_seq()

```
DDS_ReturnCode_t DDS_DynamicData::set_ulonglong_seq (
            const char * member_name,
             DDS_DynamicDataMemberId member_id,
            const  DDS_UnsignedLongLongSeq & value )
```

Set the contents of the given sequence member.

This method will perform an automatic conversion to an array of **DDS_UnsignedLongLong** (p. 515).

This API is not supported when the DynamicData object is in CDR format and will fail with **DDS_RETCODE_↩ PRECONDITION_NOT_MET** (p. 532). See **DDS_DynamicData::is_cdr** (p. 1035) for more information.

MT Safety:

UNSAFE.

Parameters

member_name	<<***in***>> *(p. 433)* The name of the member or NULL to look up the member by its ID.
member_id	<<***in***>> *(p. 433)* The ID of the member or **DDS_DYNAMIC_DATA_MEMBER_ID_UNSPECIFIED** (p. 105) to look up by name. See **Member Names and IDs** (p. 1022).
value	<<***in***>> *(p. 433)* A sequence, from which the elements will be copied.

Exceptions

One	of the **Standard Return Codes** (p. 532) or **DDS_RETCODE_OUT_OF_RESOURCES** (p. 533)

See also

DDS_DynamicData::get_ulonglong_seq (p. 1089)

DDS_DynamicData::set_ulonglong_array (p. 1115)

9.58.6.125 set_longdouble_seq()

```
DDS_ReturnCode_t DDS_DynamicData::set_longdouble_seq (
            const char * member_name,
             DDS_DynamicDataMemberId member_id,
            const  DDS_LongDoubleSeq & value )
```

Set the contents of the given sequence member.

This method will perform an automatic conversion to an array of **DDS_LongDouble** (p. 516).

This API is not supported when the DynamicData object is in CDR format and will fail with **DDS_RETCODE_↩ PRECONDITION_NOT_MET** (p. 532). See **DDS_DynamicData::is_cdr** (p. 1035) for more information.

MT Safety:

UNSAFE.

Parameters

member_name	<<***in***>> *(p. 433)* The name of the member or NULL to look up the member by its ID.
member_id	<<***in***>> *(p. 433)* The ID of the member or **DDS_DYNAMIC_DATA_MEMBER_ID_UNSPECIFIED** (p. 105) to look up by name. See **Member Names and IDs** (p. 1022).
value	<<***in***>> *(p. 433)* A sequence, from which the elements will be copied.

Exceptions

One	of the **Standard Return Codes** (p. 532) or **DDS_RETCODE_OUT_OF_RESOURCES** (p. 533)

See also

DDS_DynamicData::get_longdouble_seq (p. 1090)

DDS_DynamicData::set_longdouble_array (p. 1116)

9.58.6.126 set_wchar_seq()

```
DDS_ReturnCode_t DDS_DynamicData::set_wchar_seq (
            const char * member_name,
             DDS_DynamicDataMemberId member_id,
            const  DDS_WcharSeq & value )
```

Set the contents of the given sequence member.

This method will perform an automatic conversion to an array of **DDS_Wchar** (p. 514).

This API is not supported when the DynamicData object is in CDR format and will fail with **DDS_RETCODE_↩ PRECONDITION_NOT_MET** (p. 532). See **DDS_DynamicData::is_cdr** (p. 1035) for more information.

MT Safety:

UNSAFE.

Parameters

member_name	*<<**in**>> (p. 433)* The name of the member or NULL to look up the member by its ID.
member_id	*<<**in**>> (p. 433)* The ID of the member or **DDS_DYNAMIC_DATA_MEMBER_ID_UNSPECIFIED** (p. 105) to look up by name. See **Member Names and IDs** (p. 1022).
value	*<<**in**>> (p. 433)* A sequence, from which the elements will be copied.

Exceptions

One	of the **Standard Return Codes** (p. 532) or **DDS_RETCODE_OUT_OF_RESOURCES** (p. 533)

See also

DDS_DynamicData::get_wchar_seq (p. 1091)

DDS_DynamicData::set_wchar_array (p. 1117)

9.58.6.127 set_int8_seq()

```
DDS_ReturnCode_t DDS_DynamicData::set_int8_seq (
            const char * member_name,
             DDS_DynamicDataMemberId member_id,
            const  DDS_Int8Seq & value )
```

Set the contents of the given sequence member.

This method will perform an automatic conversion to an array of **DDS_Int8** (p. 514).

This API is not supported when the DynamicData object is in CDR format and will fail with **DDS_RETCODE_↩ PRECONDITION_NOT_MET** (p. 532). See **DDS_DynamicData::is_cdr** (p. 1035) for more information.

MT Safety:

UNSAFE.

Parameters

member_name	<<***in***>> *(p. 433)* The name of the member or NULL to look up the member by its ID.
member_id	<<***in***>> *(p. 433)* The ID of the member or **DDS_DYNAMIC_DATA_MEMBER_ID_UNSPECIFIED** (p. 105) to look up by name. See **Member Names and IDs** (p. 1022).
value	<<***in***>> *(p. 433)* A sequence, from which the elements will be copied.

Exceptions

One	of the **Standard Return Codes** (p. 532) or **DDS_RETCODE_OUT_OF_RESOURCES** (p. 533)

See also

DDS_DynamicData::get_int8_seq (p. 1092)

DDS_DynamicData::set_int8_array (p. 1118)

9.58.6.128 set_uint8_seq()

```
DDS_ReturnCode_t DDS_DynamicData::set_uint8_seq (
            const char * member_name,
             DDS_DynamicDataMemberId member_id,
            const  DDS_UInt8Seq & value )
```

Set the contents of the given sequence member.

This method will perform an automatic conversion to an array of **DDS_UInt8** (p. 514).

This API is not supported when the DynamicData object is in CDR format and will fail with **DDS_RETCODE_↩ PRECONDITION_NOT_MET** (p. 532). See **DDS_DynamicData::is_cdr** (p. 1035) for more information.

MT Safety:

UNSAFE.

Parameters

member_name	<<***in***>> *(p. 433)* The name of the member or NULL to look up the member by its ID.
member_id	<<***in***>> *(p. 433)* The ID of the member or **DDS_DYNAMIC_DATA_MEMBER_ID_UNSPECIFIED** (p. 105) to look up by name. See **Member Names and IDs** (p. 1022).
value	<<***in***>> *(p. 433)* A sequence, from which the elements will be copied.

Exceptions

One	of the **Standard Return Codes** (p. 532) or **DDS_RETCODE_OUT_OF_RESOURCES** (p. 533)

See also

DDS_DynamicData::get_uint8_seq (p. 1092)

DDS_DynamicData::set_uint8_array (p. 1119)

9.58.6.129 set_discriminator()

```
DDS_ReturnCode_t DDS_DynamicData::set_discriminator (
            DDS_Long discriminator )
```

Set the value of the union discriminator (UnionType only)

This operation is valid only for objects of **DDS_TCKind** (p. 89) **DDS_TK_UNION** (p. 89).

If the discriminator value identifies the same member in the union, the previously set value for the member is preserved and only the discriminator value is updated.

If the discriminator value identifies a different member in the union, the previously set member in the union will be finalized and the identified member will be initialized to its default value.

If the discriminator doesn't identify any member in the union, then the previously set member will be finalized and the discriminator will be set to the provided value.

For example, given the following IDL Union:

```
union Foo switch (short) {
case 1:
    long x;
case 2:
case 3:
    string y;
};
```

We can set the active member to "y" as follows (all code omits error checking for brevity):

```
// Set the discriminator to 2 and select member "y"
DDS_DynamicData_set_discriminator(sample, 2);

DDS_DynamicData_get_string(
        sample,
        &outString,
        size,
        "y",
        DDS_DYNAMIC_DATA_MEMBER_ID_UNSPECIFIED);
assert(outString == "");

DDS_DynamicData_set_string(
        sample,
        "y",
        DDS_DYNAMIC_DATA_MEMBER_ID_UNSPECIFIED,
        "hello");
```

Setting the discriminator to 3 also selects "y" and preserves the previous value of "hello".

```
DDS_DynamicData_set_discriminator(sample, 3);
```

```
DDS_DynamicData_get_string(
        sample,
        &outString,
        size,
        "y",
        DDS_DYNAMIC_DATA_MEMBER_ID_UNSPECIFIED);
assert(outString == "hello");
```

Setting the discriminator to a new value that selects a different member, "x", finalizes the previous member's value and initializes the new member to its default.

```
DDS_DynamicData_set_discriminator(sample, 1);

DDS_DynamicData_get_long(
        sample,
        &outLong,
        "x",
        DDS_DYNAMIC_DATA_MEMBER_ID_UNSPECIFIED);
assert(outLong == 0);

retCode = DDS_DynamicData_get_string(
        sample,
        &outString,
        size,
        "y",
        DDS_DYNAMIC_DATA_MEMBER_ID_UNSPECIFIED);
assert(retCode == DDS_RETCODE_NO_DATA);
```

The discriminator can also be set to a value that does not select any member.

```
DDS_DynamicData_set_discriminator(sample, 100);
DDS_DynamicData_get_discriminator(sample, &discValue);
assert(discValue == 100);
assert(DDS_DynamicData_get_member_count(sample) == 0);
```

This API is not supported when the DynamicData object is in CDR format and will fail with **DDS_RETCODE_↩ PRECONDITION_NOT_MET** (p. 532). See **DDS_DynamicData::is_cdr** (p. 1035) for more information.

MT Safety:

UNSAFE.

Parameters

discriminator	<<***in***>> *(p. 433)* The discriminator value to set

Exceptions

One	of the **Standard Return Codes** (p. 532)

See also

DDS_DynamicData::get_discriminator (p. 1067)

Getting and Setting Union Discriminator Values (p. 1022)

DDS_DynamicData::get_member_count (p. 1045)

9.59 DDS_DynamicDataInfo Struct Reference

A descriptor for a **DDS_DynamicData** (p. 1013) object.

Public Attributes

- **DDS_Long member_count**

  *The number of data members in this **DDS_DynamicData** (p. 1013) sample.*
- **DDS_Long stored_size**

  *The number of bytes currently used to store the data of this **DDS_DynamicData** (p. 1013) sample.*

9.59.1 Detailed Description

A descriptor for a **DDS_DynamicData** (p. 1013) object.

See also

DDS_DynamicData::get_info (p. 1040)

9.59.2 Member Data Documentation

9.59.2.1 member_count

```
DDS_Long DDS_DynamicDataInfo::member_count
```

The number of data members in this **DDS_DynamicData** (p. 1013) sample.

9.59.2.2 stored_size

```
DDS_Long DDS_DynamicDataInfo::stored_size
```

The number of bytes currently used to store the data of this **DDS_DynamicData** (p. 1013) sample.

9.60 DDS_DynamicDataJsonParserProperties_t Struct Reference

A collection of attributes used to configure **DDS_DynamicData** (p. 1013) objects.

9.60.1 Detailed Description

A collection of attributes used to configure **DDS_DynamicData** (p. 1013) objects.

9.61 DDS_DynamicDataMemberInfo Struct Reference

A descriptor for a single member (i.e. field) of dynamically defined data type.

Public Attributes

- **DDS_DynamicDataMemberId member_id**

  *An integer that uniquely identifies the data member within this **DDS_DynamicData** (p. 1013) sample's type.*
- const char * **member_name**

  The string name of the data member.
- **DDS_Boolean member_exists**

  Indicates whether the member exists in this sample.
- **DDS_TCKind member_kind**

  The kind of type of this data member (e.g., integer, structure, etc.).
- **DDS_UnsignedLong element_count**

  The number of elements within this data member.
- **DDS_TCKind element_kind**

  The kind of type of the elements within this data member.

9.61.1 Detailed Description

A descriptor for a single member (i.e. field) of dynamically defined data type.

See also

DDS_DynamicData::get_member_info (p. 1047)

9.61.2 Member Data Documentation

9.61.2.1 member_id

DDS_DynamicDataMemberId DDS_DynamicDataMemberInfo::member_id

An integer that uniquely identifies the data member within this **DDS_DynamicData** (p. 1013) sample's type.

The member ID is assigned automatically by the middleware based on the member's declaration order within the type. It is a 1-based index of the member's position in the type.

See also

DDS_TCKind (p. 89)

9.61.2.2 member_name

```
const char* DDS_DynamicDataMemberInfo::member_name
```

The string name of the data member.

This name will be unique among members of the same type. However, a single named member may have multiple type representations.

9.61.2.3 member_exists

```
DDS_Boolean DDS_DynamicDataMemberInfo::member_exists
```

Indicates whether the member exists in this sample.

A member that is defined in a type may not exist in a sample of that type in the following situations

- The member is optional and is unset
- Being part of a union, the discriminator doesn't select this member
- When getting the information about a sequence element, with index `i`, and `i` is greater than the current sequence length but smaller than its maximum length.

See also

DDS_DynamicData::member_exists_in_type (p. 1047)

9.61.2.4 member_kind

```
DDS_TCKind DDS_DynamicDataMemberInfo::member_kind
```

The kind of type of this data member (e.g., integer, structure, etc.).

This is a convenience field; it is equivalent to looking up the member in the **DDS_TypeCode** (p. 1428) and getting the **DDS_TCKind** (p. 89) from there.

9.61.2.5 element_count

```
DDS_UnsignedLong DDS_DynamicDataMemberInfo::element_count
```

The number of elements within this data member.

This information is only valid for members of array or sequence types. Members of other types will always report zero (0) here.

9.61.2.6 element_kind

DDS_TCKind DDS_DynamicDataMemberInfo::element_kind

The kind of type of the elements within this data member.

This information is only valid for members of array or sequence types. Members of other types will always report **DDS_TK_NULL** (p. 89) here.

9.62 DDS_DynamicDataProperty_t Struct Reference

A collection of attributes used to configure **DDS_DynamicData** (p. 1013) objects.

Public Attributes

- **DDS_Long buffer_initial_size**

  *The initial amount of memory used by the underlying **DDS_DynamicData** (p. 1013) buffer, in bytes.*
- **DDS_Long buffer_max_size**

  *The maximum amount of memory that the underlying **DDS_DynamicData** (p. 1013) buffer may use, in bytes.*

9.62.1 Detailed Description

A collection of attributes used to configure **DDS_DynamicData** (p. 1013) objects.

9.62.2 Member Data Documentation

9.62.2.1 buffer_initial_size

DDS_Long DDS_DynamicDataProperty_t::buffer_initial_size

The initial amount of memory used by the underlying **DDS_DynamicData** (p. 1013) buffer, in bytes.

This property is used to configure the DynamicData objects that are created stand-alone as well as the DynamicData samples that are obtained from the sample pool that is created by each DynamicData DataReader.

If set to 0 (default): The initial buffer size will be set to the minimum amount of space required to hold the overhead required by the DynamicData internal representation (about 100 bytes) in addition to the minimum deserialized size of a sample. The minimum deserialized size of a sample assumes that all strings are allocated to their default values, sequences are left to length 0, and all optional members are unset.

If set to any value other than 0: The underlying buffer will be allocated to the provided size plus the overhead required by the DynamicData internal representation (about 100 bytes). If the provided size plus the overhead is less than the size used when buffer_initial_size is left to 0, then the default value is used.

See also

DDS_DynamicDataProperty_t::buffer_max_size (p. 1137)

9.62.2.2 buffer_max_size

```
DDS_Long DDS_DynamicDataProperty_t::buffer_max_size
```

The maximum amount of memory that the underlying **DDS_DynamicData** (p. 1013) buffer may use, in bytes.

This property is used to configure the DynamicData objects that are created stand-alone as well as the DynamicData samples that are obtained from the sample pool that is created by each DynamicData DataReader.

A DynamicData object will grow to this size from the initial size as needed. The buffer_max_size includes all overhead that is required for the internal DynamicData representation and therefore represents a hard upper limit on the size of the underlying DynamicData buffer.

If set to -1 (default): The buffer will grow unbounded to the size required to fit all members.

If set to any value other than -1: The buffer will not grow beyond this size. If setting a member's values requires the buffer to grow beyond this maximum, the member will fail to be set. If the buffer is required to grow beyond this maximum during deserialization, the sample will fail to be deserialized. The buffer_max_size cannot be smaller than the buffer_initial_size.

See also

DDS_DynamicDataProperty_t::buffer_initial_size (p. 1137)

9.63 DDS_DynamicDataSeq Struct Reference

An ordered collection of **DDS_DynamicData** (p. 1013) elements.

9.63.1 Detailed Description

An ordered collection of **DDS_DynamicData** (p. 1013) elements.

Instantiates **FooSeq** (p. 1965) < **DDS_DynamicData** (p. 1013) > .

See also

FooSeq (p. 1965)

DDS_DynamicData (p. 1013)

9.64 DDS_DynamicDataTypeProperty_t Struct Reference

A collection of attributes used to configure **DDSDynamicDataTypeSupport** (p. 1725) objects.

Public Attributes

- struct **DDS_DynamicDataProperty_t data**

  *These properties will be provided to every new **DDS_DynamicData** (p. 1013) sample created from the **DDSDynamic↩ DataTypeSupport** (p. 1725).*
- struct **DDS_DynamicDataTypeSerializationProperty_t serialization**

  Properties that govern how the data of this type will be serialized on the network.

9.64.1 Detailed Description

A collection of attributes used to configure **DDSDynamicDataTypeSupport** (p. 1725) objects.

The properties of a **DDSDynamicDataTypeSupport** (p. 1725) object contain the properties that will be used to instantiate any samples created by that object.

9.64.2 Member Data Documentation

9.64.2.1 data

`struct` **`DDS_DynamicDataProperty_t`** `DDS_DynamicDataTypeProperty_t::data`

These properties will be provided to every new **DDS_DynamicData** (p. 1013) sample created from the **DDSDynamic↩ DataTypeSupport** (p. 1725).

These properties are also used to create the samples that are kept in the DataReader's queue.

9.64.2.2 serialization

`struct` **`DDS_DynamicDataTypeSerializationProperty_t`** `DDS_DynamicDataTypeProperty_t::serialization`

Properties that govern how the data of this type will be serialized on the network.

9.65 DDS_DynamicDataTypeSerializationProperty_t Struct Reference

Properties that govern how data of a certain type will be serialized on the network.

Public Attributes

- **DDS_Boolean use_42e_compatible_alignment**

  *[**No longer supported**] Use RTI Connext 4.2e-compatible alignment for large primitive types.*
- **DDS_UnsignedLong max_size_serialized**

  *[**No longer supported**] If you were previously using this property, use **DDS_DynamicDataProperty_t::buffer_max_size** (p. 1137) instead. The maximum number of bytes that objects of a given type could consume when serialized on the network.*
- **DDS_UnsignedLong min_size_serialized**

  *[**No longer supported**] If you were previously using this property, use **DDS_DynamicDataProperty_t::buffer_initial_size** (p. 1137) instead. The minimum number of bytes that objects of a given type could consume when serialized on the network.*
- **DDS_Boolean trim_to_size**

  Controls the growth of the buffer in a DynamicData object.
- **DDS_Boolean skip_deserialization**

  Enable skipping the deserialization of data on the DataReader side.

9.65.1 Detailed Description

Properties that govern how data of a certain type will be serialized on the network.

9.65.2 Member Data Documentation

9.65.2.1 use_42e_compatible_alignment

```
DDS_Boolean DDS_DynamicDataTypeSerializationProperty_t::use_42e_compatible_alignment
```

[**No longer supported**] Use RTI Connext 4.2e-compatible alignment for large primitive types.

In RTI Connext 4.2e, the default alignment for large primitive types – **DDS_LongLong** (p. 515), **DDS_UnsignedLong Long** (p. 515), **DDS_Double** (p. 515), and **DDS_LongDouble** (p. 516) – was not RTPS-compliant. This compatibility mode allows applications targeting post-4.2e versions of RTI Connext to interoperate with 4.2e-based applications, regardless of the data types they use.

If this flag is not set, all data will be serialized in an RTPS-compliant manner, which for the types listed above, will not be interoperable with RTI Connext 4.2e.

9.65.2.2 max_size_serialized

```
DDS_UnsignedLong DDS_DynamicDataTypeSerializationProperty_t::max_size_serialized
```

[**No longer supported**] If you were previously using this property, use **DDS_DynamicDataProperty_t::buffer_max_size** (p. 1137) instead. The maximum number of bytes that objects of a given type could consume when serialized on the network.

This value is used to set the sizes of certain internal middleware buffers.

The effective value of the maximum serialized size will be the value of this field or the size automatically inferred from the type's **DDS_TypeCode** (p. 1428), whichever is smaller.

9.65.2.3 min_size_serialized

```
DDS_UnsignedLong DDS_DynamicDataTypeSerializationProperty_t::min_size_serialized
```

[**No longer supported**] If you were previously using this property, use **DDS_DynamicDataProperty_t::buffer_initial_size** (p. 1137) instead. The minimum number of bytes that objects of a given type could consume when serialized on the network.

This value is used to set the sizes of certain internal middleware buffers.

Default: 0xFFFFFFFF, a sentinel that indicates that this value must be equal to the value specified in max_size_serialized.

9.65.2.4 trim_to_size

```
DDS_Boolean DDS_DynamicDataTypeSerializationProperty_t::trim_to_size
```

Controls the growth of the buffer in a DynamicData object.

This property only applies to DynamicData samples that are obtained from the sample pool that is created by each DynamicData DataReader.

If set to 0 (default): The buffer will not be reallocated unless the deserialized size of the incoming sample is greater than the current buffer size.

If set to 1: The buffer of a DynamicData object obtained from the DDS sample pool will be freed and re-allocated for each sample to just fit the size of the deserialized data of the incoming sample. The newly allocated size will not be smaller than **DDS_DynamicDataProperty_t::buffer_initial_size** (p. 1137).

9.65.2.5 skip_deserialization

```
DDS_Boolean DDS_DynamicDataTypeSerializationProperty_t::skip_deserialization
```

Enable skipping the deserialization of data on the DataReader side.

Setting this member to **DDS_BOOLEAN_TRUE** (p. 513) boosts the performance for use cases in which the application does not require accessing the value of the different members of the received DynamicData object. Those use cases include recording, replaying, and bridging. You can access the CDR buffer of a DynamicData object by calling the method **DDS_DynamicData::get_cdr_buffer** (p. 1035). This configuration option is only available in C, Traditional C++, Modern C++, and Python language bindings.

When this option is set to **DDS_BOOLEAN_TRUE** (p. 513), many of the DynamicData APIs cannot be used to access or manipulate the received DynamicData object. See each functions's documentation for whether or not the function is supported for DynamicData objects in CDR format.

See also

DDS_DynamicData::is_cdr (p. 1035)

DDS_DynamicData::get_cdr_buffer (p. 1035)

DDS_DynamicData::set_cdr_buffer (p. 1036)

[default] DDS_BOOLEAN_FALSE (p. 513)

9.66 DDS_EndpointGroup_t Struct Reference

Specifies a group of endpoints that can be collectively identified by a name and satisfied by a quorum.

Public Attributes

- char * **role_name**

  Defines the role name of the endpoint group.
- int **quorum_count**

  Defines the minimum number of members that satisfies the endpoint group.

9.66.1 Detailed Description

Specifies a group of endpoints that can be collectively identified by a name and satisfied by a quorum.

9.66.2 Member Data Documentation

9.66.2.1 role_name

```
char* DDS_EndpointGroup_t::role_name
```

Defines the role name of the endpoint group.

If used in the **DDS_AvailabilityQosPolicy** (p. 805) on a **DDSDataWriter** (p. 1589), it specifies the name that identifies a Durable Subcription.

The role name can be at most 255 characters in length.

9.66.2.2 quorum_count

```
int DDS_EndpointGroup_t::quorum_count
```

Defines the minimum number of members that satisfies the endpoint group.

If used in the **DDS_AvailabilityQosPolicy** (p. 805) on a **DDSDataWriter** (p. 1589), it specifies the number of Data↩ Readers that must acknowledge a sample before the sample is considered to be acknowledged by the Durable Subscription.

9.67 DDS_EndpointGroupSeq Struct Reference

A sequence of **DDS_EndpointGroup_t** (p. 1142).

9.67.1 Detailed Description

A sequence of **DDS_EndpointGroup_t** (p. 1142).

In the context of Collaborative DataWriters, it can be used by a **DDSDataReader** (p. 1556) to define a group of remote DataWriters that the **DDSDataReader** (p. 1556) will wait to discover before skipping missing samples.

In the context of Durable Subscriptions, it can be used to create a set of Durable Subscriptions identified by a name and a quorum count.

Instantiates:

<<***generic***>> *(*p. *433)* **FooSeq** (p. 1965)

See also

DDS_EndpointGroup_t (p. 1142)

9.68 DDS_EndpointTrustAlgorithmInfo Struct Reference

Trust Plugins algorithm information associated with the discovered endpoint.

Public Attributes

- **DDS_EndpointTrustInterceptorAlgorithmInfo interceptor**
  Information regarding algorithms for interception of data and metadata exchanged by the endpoint.

9.68.1 Detailed Description

Trust Plugins algorithm information associated with the discovered endpoint.

9.68.2 Member Data Documentation

9.68.2.1 interceptor

DDS_EndpointTrustInterceptorAlgorithmInfo DDS_EndpointTrustAlgorithmInfo::interceptor

Information regarding algorithms for interception of data and metadata exchanged by the endpoint.

9.69 DDS_EndpointTrustInterceptorAlgorithmInfo Struct Reference

Trust Plugins interception algorithm information associated with the discovered endpoint.

Public Attributes

- **DDS_TrustAlgorithmSet required_mask**

  Trust Plugins algorithms used for interception of data and metadata exchanged by the endpoint.
- **DDS_TrustAlgorithmSet supported_mask**

  Trust Plugins algorithms supported for interception of data and metadata exchanged by the endpoint.

9.69.1 Detailed Description

Trust Plugins interception algorithm information associated with the discovered endpoint.

9.69.2 Member Data Documentation

9.69.2.1 required_mask

DDS_TrustAlgorithmSet DDS_EndpointTrustInterceptorAlgorithmInfo::required_mask

Trust Plugins algorithms used for interception of data and metadata exchanged by the endpoint.

9.69.2.2 supported_mask

DDS_TrustAlgorithmSet DDS_EndpointTrustInterceptorAlgorithmInfo::supported_mask

Trust Plugins algorithms supported for interception of data and metadata exchanged by the endpoint.

9.70 DDS_EndpointTrustProtectionInfo Struct Reference

Trust Plugins Protection information associated with the discovered endpoint.

Public Attributes

- **DDS_EndpointTrustAttributesMask bitmask**

  Information about how RTPS wire serialization, discovery, and liveliness interact with the associated Trust Plugins Protection configuration.
- **DDS_PluginEndpointTrustAttributesMask plugin_bitmask**

  Internal plugin information that is opaque to DDS.

9.70.1 Detailed Description

Trust Plugins Protection information associated with the discovered endpoint.

9.70.2 Member Data Documentation

9.70.2.1 bitmask

DDS_EndpointTrustAttributesMask DDS_EndpointTrustProtectionInfo::bitmask

Information about how RTPS wire serialization, discovery, and liveliness interact with the associated Trust Plugins Protection configuration.

The meaning of the contents of this field may vary depending on what Trust Plugins the endpoint is using. For information about how this field interacts with the RTI Security Plugins, please refer to the `RTI Security Plugins User's Manual`.

9.70.2.2 plugin_bitmask

DDS_PluginEndpointTrustAttributesMask DDS_EndpointTrustProtectionInfo::plugin_bitmask

Internal plugin information that is opaque to DDS.

The meaning of the contents of this field may vary depending on what Trust Plugins the endpoint is using. For information about how this field interacts with the RTI Security Plugins, please refer to the `RTI Security Plugins User's Manual`.

9.71 DDS_EntityFactoryQosPolicy Struct Reference

<<***spec***>> *(p. 209)* A QoS policy for all **DDSEntity** (p. 1731) types that can act as factories for one or more other **DDSEntity** (p. 1731) types.

Public Attributes

- **DDS_Boolean autoenable_created_entities**

  *<<**basic**>> (p. 201) <<**spec**>> (p. 209) Specifies whether the entity acting as a factory automatically enables the instances it creates.*

9.71.1 Detailed Description

*<<**spec**>> (p. 209)* A QoS policy for all **DDSEntity** (p. 1731) types that can act as factories for one or more other **DDSEntity** (p. 1731) types.

Entity:

DDSDomainParticipantFactory (p. 1694), **DDSDomainParticipant** (p. 1619), **DDSPublisher** (p. 1818), **DDSSubscriber** (p. 1860)

Properties:

RxO (p. 556) = NO
Changeable (p. 556) = **YES** (p. 556)

9.71.2 Usage

This policy controls the behavior of the **DDSEntity** (p. 1731) as a factory for other entities. It controls whether or not child entities are created in the enabled state.

RTI Connext uses a factory design pattern for creating DDS Entities. That is, a parent entity must be used to create child entities. DomainParticipants create Topics, Publishers and Subscribers. Publishers create DataWriters. Subscribers create DataReaders.

By default, a child object is enabled upon creation (initialized and may be actively used). With this QoS policy, a child object can be created in a disabled state. A disabled entity is only partially initialized and cannot be used until the entity is enabled. Note: an entity can only be *enabled*; it cannot be *disabled* after it has been enabled.

This QoS policy is useful to synchronize the initialization of DDS Entities. For example, when a **DDSDataReader** (p. 1556) is created in an enabled state, its existence is immediately propagated for discovery and the **DDSDataReader** (p. 1556) object's listener called as soon as data is received. The initialization process for an application may extend beyond the creation of the **DDSDataReader** (p. 1556), and thus, it may not be desireable for the **DDSDataReader** (p. 1556) to start to receive or process any data until the initialization process is complete. So by creating readers in a disabled state, your application can make sure that no data is received until the rest of the application initialization is complete, and at that time, enable the them.

Note: if an entity is disabled, then all of the child entities it creates will be disabled, too, regardless of the setting of this QoS policy. However, enabling a disabled entity will enable all of its children if this QoS policy is set to automatically enable children entities.

This policy is mutable. A change in the policy affects only the entities created after the change, not any previously created entities.

9.71.3 Member Data Documentation

9.71.3.1 autoenable_created_entities

```
DDS_Boolean DDS_EntityFactoryQosPolicy::autoenable_created_entities
```

<<***basic***>> (p. *201)* <<***spec***>> (p. *209)* Specifies whether the entity acting as a factory automatically enables the instances it creates.

The setting of `autoenable_created_entities` to **DDS_BOOLEAN_TRUE** (p. 513) indicates that the factory `create_<entity>` operation(s) will automatically invoke the **DDSEntity::enable** (p. 1734) operation each time a new **DDSEntity** (p. 1731) is created. Therefore, the **DDSEntity** (p. 1731) returned by `create_<entity>` will already be enabled. A setting of **DDS_BOOLEAN_FALSE** (p. 513) indicates that the **DDSEntity** (p. 1731) will not be automatically enabled. Your application will need to call **DDSEntity::enable** (p. 1734) itself.

The default setting of `autoenable_created_entities` = **DDS_BOOLEAN_TRUE** (p. 513) means that, by default, it is not necessary to explicitly call **DDSEntity::enable** (p. 1734) on newly created entities.

[default] DDS_BOOLEAN_TRUE (p. 513)

Categories

Mutable (p. 369), **Application Startup** (p. 256)

9.72 DDS_EntityNameQosPolicy Struct Reference

<<***extension***>> (p. *213)* Assigns a name and a role name to a **DDSDomainParticipant** (p. 1619), **DDSPublisher** (p. 1818), **DDSSubscriber** (p. 1860), **DDSDataWriter** (p. 1589) or **DDSDataReader** (p. 1556). Except for **DDSPublisher** (p. 1818) and **DDSSubscriber** (p. 1860), these names will be visible during the discovery process and in RTI tools to help you visualize and debug your system.

Public Attributes

- char * **name**

  <<***basic***>> (p. *201)* <<***extension***>> (p. *213) The name of the entity.*
- char * **role_name**

  <<***extension***>> (p. *213) The entity role name.*

9.72.1 Detailed Description

<<***extension***>> *(p. 213)* Assigns a name and a role name to a **DDSDomainParticipant** (p. 1619), **DDSPublisher** (p. 1818), **DDSSubscriber** (p. 1860), **DDSDataWriter** (p. 1589) or **DDSDataReader** (p. 1556). Except for **DDSPublisher** (p. 1818) and **DDSSubscriber** (p. 1860), these names will be visible during the discovery process and in RTI tools to help you visualize and debug your system.

Entity:

DDSDomainParticipant (p. 1619), **DDSSubscriber** (p. 1860), **DDSPublisher** (p. 1818), **DDSDataReader** (p. 1556), **DDSDataWriter** (p. 1589)

Properties:

RxO (p. 556) = NO;
Changeable (p. 556) = **UNTIL ENABLE** (p. 556)

9.72.2 Usage

The name and role name can be at most 255 characters in length.

The strings must be null-terminated strings allocated with **DDS_String_alloc** (p. 759) or **DDS_String_dup** (p. 761).

If you provide a non-null pointer when getting the QoS, then it should point to valid memory that can be written to, to avoid ungraceful failures.

9.72.3 Member Data Documentation

9.72.3.1 name

```
char* DDS_EntityNameQosPolicy::name
```

<<***basic***>> *(p. 201)* <<***extension***>> *(p. 213)* The name of the entity.

[default] null

[range] Null terminated string with length not exceeding 255. It can be null.

Categories

Immutable (p. 333), **Debuggability** (p. 266), **Discovery Metadata** (p. 377)

9.72.3.2 role_name

```
char* DDS_EntityNameQosPolicy::role_name
```

<<***extension***>> *(p. 213)* The entity role name.

With Durable Subscriptions this name is used to specify to which Durable Subscription the **DDSDataReader** (p. 1556) belongs.

With Collaborative DataWriters this name is used to specify to which endpoint group the **DDSDataWriter** (p. 1589) belongs.

[range] Null terminated string with length not exceeding 255. It can be null.

[default] null

Categories

Immutable (p. 333), **Debuggability** (p. 266), **Discovery Metadata** (p. 377), **Durability** (p. 278)

9.73 DDS_EnumMember Struct Reference

A description of a member of an enumeration.

Public Attributes

- char * **name**

  The name of the enumeration member.
- **DDS_Long ordinal**

  The value associated the the enumeration member.

9.73.1 Detailed Description

A description of a member of an enumeration.

See also

DDS_EnumMemberSeq (p. 1150)

DDS_TypeCodeFactory::create_enum_tc (p. 1485)

9.73.2 Member Data Documentation

9.73.2.1 name

```
char* DDS_EnumMember::name
```

The name of the enumeration member.

Cannot be NULL.

9.73.2.2 ordinal

```
DDS_Long DDS_EnumMember::ordinal
```

The value associated the the enumeration member.

9.74 DDS_EnumMemberSeq Struct Reference

Defines a sequence of enumerator members.

9.74.1 Detailed Description

Defines a sequence of enumerator members.

See also

DDS_EnumMember (p. 1149)
FooSeq (p. 1965)
DDS_TypeCodeFactory::create_enum_tc (p. 1485)

9.75 DDS_EventQosPolicy Struct Reference

<<***extension***>> *(p. 213)* Settings for event.

Public Attributes

- struct **DDS_ThreadSettings_t thread**
  <<***extension***>> *(p. 213) Event thread QoS.*
- **DDS_Long initial_count**
  <<***extension***>> *(p. 213) The initial number of events.*
- **DDS_Long max_count**
  <<***extension***>> *(p. 213) The maximum number of events.*

9.75.1 Detailed Description

<<***extension***>> *(p. 213)* Settings for event.

In a **DDSDomainParticipant** (p. 1619), a thread is dedicated to handle all timed events, including checking for timeouts and deadlines and executing internal and user-defined timeout or exception handling routines/callbacks.

This QoS policy allows you to configure thread properties such as priority level and stack size. You can also configure the maximum number of events that can be posted to the event thread. By default, a **DDSDomainParticipant** (p. 1619) will dynamically allocate memory as needed for events posted to the event thread. However, by setting a maximum value or setting the initial and maximum value to be the same, you can either bound the amount of memory allocated for the event thread or prevent a **DDSDomainParticipant** (p. 1619) from dynamically allocating memory for the event thread after initialization.

This QoS policy is an extension to the DDS standard.

Entity:

DDSDomainParticipant (p. 1619)

Properties:

RxO (p. 556) = N/A
Changeable (p. 556) = **NO** (p. 556)

9.75.2 Member Data Documentation

9.75.2.1 thread

`struct` **`DDS_ThreadSettings_t`** `DDS_EventQosPolicy::thread`

<<***extension***>> *(p. 213)* Event thread QoS.

There is only one event thread.

Priority:
[default] The actual value depends on your architecture:
For Windows: -2
For Linux: OS default priority
For a complete list of platform specific values, please refer to `Platform Notes`.

Stack Size:
[default] The actual value depends on your architecture:
For Windows: OS default stack size
For Linux: OS default stack size
For a complete list of platform specific values, please refer to `Platform Notes`.

Mask:
[default] mask = **DDS_THREAD_SETTINGS_FLOATING_POINT** (p. 548) | **DDS_THREAD_SETTINGS_STDIO** (p. 548)

Categories

Immutable (p. 333), **Concurrency** (p. 258)

9.75.2.2 initial_count

```
DDS_Long DDS_EventQosPolicy::initial_count
```

<<***extension***>> *(p. 213)* The initial number of events.

[default] 256

[range] [1, 1 million], <= max_count

Categories

Immutable (p. 333), **Middleware Memory Management** (p. 297)

9.75.2.3 max_count

```
DDS_Long DDS_EventQosPolicy::max_count
```

<<***extension***>> *(p. 213)* The maximum number of events.

The maximum number of events. If the limit is reached, no new event can be added.

[default] DDS_LENGTH_UNLIMITED (p. 636)

[range] [1, 1 million] or **DDS_LENGTH_UNLIMITED** (p. 636), >= initial_count

Categories

Immutable (p. 333), **Middleware Memory Management** (p. 297)

9.76 DDS_ExpressionProperty Struct Reference

Provides additional information about the filter expression passed to **DDSWriterContentFilter::writer_compile** (p. 1906) .

Public Attributes

- **DDS_Boolean key_only_filter**

  Indicates if the filter expression is based only on key fields. In this case, RTI Connext itself can cache the filtering results.
- **DDS_Boolean writer_side_filter_optimization**

  Indicates if the filter implementation can cache the filtering result for the provided expression.

9.76.1 Detailed Description

Provides additional information about the filter expression passed to **DDSWriterContentFilter::writer_compile** (p. 1906) .

It is used by the filter implementation to indicate to the middleware whether or not the **DDSWriterContentFilter** (p. 1905) will cache the result of filter evaluation.

9.76.2 Member Data Documentation

9.76.2.1 key_only_filter

```
DDS_Boolean DDS_ExpressionProperty::key_only_filter
```

Indicates if the filter expression is based only on key fields. In this case, RTI Connext itself can cache the filtering results.

When this field is set to **DDS_BOOLEAN_TRUE** (p. 513), it indicates to RTI Connext that the filter expression is based only on key fields.

9.76.2.2 writer_side_filter_optimization

```
DDS_Boolean DDS_ExpressionProperty::writer_side_filter_optimization
```

Indicates if the filter implementation can cache the filtering result for the provided expression.

When this field is set to **DDS_BOOLEAN_TRUE** (p. 513), RTI Connext will do no caching or explicit filter evaluation for the associated **DDSDataReader** (p. 1556). Instead, it will rely on the filter implementation to provide appropriate results.

9.77 DDS_FilterSampleInfo Struct Reference

Provides meta information associated with the sample.

Public Attributes

- struct **DDS_SampleIdentity_t related_sample_identity**

  The identity of another sample related to this one.
- struct **DDS_GUID_t related_source_guid**

  Identifies the application logical data source that is related to the sample being written.
- struct **DDS_GUID_t related_reader_guid**

  Identifies a DataReader that is logically related to the sample.
- struct **DDS_SampleIdentity_t sample_identity**

  The identity of the sample.
- struct **DDS_GUID_t source_guid**

  Identifies the application logical data source associated with the sample being written.

9.77.1 Detailed Description

Provides meta information associated with the sample.

This can be used by a content filter to perform filtering on meta data.

9.77.2 Member Data Documentation

9.77.2.1 related_sample_identity

`struct` **`DDS_SampleIdentity_t`** `DDS_FilterSampleInfo::related_sample_identity`

The identity of another sample related to this one.

The value of this field identifies another sample that is logically related to the one that is written. For example, the **DDSDataWriter** (p. 1589) created by a Replier uses this field to associate the identity of the request sample with a response sample.

The related_sample_identity can be set using the field **DDS_WriteParams_t::related_sample_identity** (p. 1520) when writing a sample using the method **FooDataWriter::write_w_params** (p. 1956).

Default value **DDS_UNKNOWN_SAMPLE_IDENTITY** (p. 676) denotes a sample not associated with any other sample.

A **DDSDataReader** (p. 1556) can inspect the related sample identity of a received sample by accessing the fields **DDS_SampleInfo::related_original_publication_virtual_guid** (p. 1352) and **DDS_SampleInfo::related_original_↩ publication_virtual_sequence_number** (p. 1353).

9.77.2.2 related_source_guid

`struct` **`DDS_GUID_t`** `DDS_FilterSampleInfo::related_source_guid`

Identifies the application logical data source that is related to the sample being written.

The related_source_guid can be set by using the field **DDS_WriteParams_t::related_source_guid** (p. 1523) when writing a sample using the method **FooDataWriter::write_w_params** (p. 1956).

Default value **DDS_GUID_UNKNOWN** (p. 527) denotes a sample as lacking a related data source.

A **DDSDataReader** (p. 1556) can inspect the related_source_guid of a received sample by accessing the field **DDS_↩ SampleInfo::related_source_guid** (p. 1353).

9.77.2.3 related_reader_guid

struct **DDS_GUID_t** DDS_FilterSampleInfo::related_reader_guid

Identifies a DataReader that is logically related to the sample.

The related_reader_guid can be set by using the field **DDS_WriteParams_t::related_reader_guid** (p. 1523) when writing a sample using the method **FooDataWriter::write_w_params** (p. 1956).

Default value **DDS_GUID_UNKNOWN** (p. 527) denotes a sample without an associated DataReader.

A **DDSDataReader** (p. 1556) can inspect the related_reader_guid of a received sample by accessing the field **DDS_SampleInfo::related_source_guid** (p. 1353).

9.77.2.4 sample_identity

struct **DDS_SampleIdentity_t** DDS_FilterSampleInfo::sample_identity

The identity of the sample.

The value of this field uniquely identifies a sample within its source DataWriter.

The identity of a sample can be set by using the field **DDS_WriteParams_t::identity** (p. 1520) when writing a sample using the method **FooDataWriter::write_w_params** (p. 1956).

Default value **DDS_UNKNOWN_SAMPLE_IDENTITY** (p. 676) can be used to let the DataWriter assign a default identity to a sample. In this case, sample_identity.writer_guid will be set to the GUID of the DataWriter, while sample_identity.sequence_number will be set to the next sequence number to be written by the DataWriter.

A **DDSDataReader** (p. 1556) can inspect the identity of a received sample by accessing the fields **DDS_SampleInfo::original_publication_virtual_guid** (p. 1352) and **DDS_SampleInfo::original_publication_virtual_sequence_number** (p. 1352).

9.77.2.5 source_guid

struct **DDS_GUID_t** DDS_FilterSampleInfo::source_guid

Identifies the application logical data source associated with the sample being written.

The source_guid can be set by using the field **DDS_WriteParams_t::source_guid** (p. 1522) when writing a sample using the method **FooDataWriter::write_w_params** (p. 1956).

Default value **DDS_GUID_UNKNOWN** (p. 527) denotes a sample without a logical data source associated with it.

A **DDSDataReader** (p. 1556) can inspect the logical data source of a sample by accessing the field **DDS_SampleInfo::source_guid** (p. 1353).

9.78 DDS_FloatSeq Struct Reference

Instantiates **FooSeq** (p. 1965) < **DDS_Float** (p. 515) >

9.78.1 Detailed Description

Instantiates **FooSeq** (p. 1965) < **DDS_Float** (p. 515) >

Instantiates:

<<***generic***>> *(*p. *433)* **FooSeq** (p. 1965)

See also

DDS_Float (p. 515)

FooSeq (p. 1965)

9.79 DDS_FlowControllerProperty_t Struct Reference

Determines the flow control characteristics of the **DDSFlowController** (p. 1736).

Public Attributes

- **DDS_FlowControllerSchedulingPolicy scheduling_policy**
  Scheduling policy.
- struct **DDS_FlowControllerTokenBucketProperty_t token_bucket**
  Settings for the token bucket.

9.79.1 Detailed Description

Determines the flow control characteristics of the **DDSFlowController** (p. 1736).

The flow control characteristics shape the network traffic by determining how often and in what order associated asynchronous **DDSDataWriter** (p. 1589) instances are serviced and how much data they are allowed to send.

Note that these settings apply directly to the **DDSFlowController** (p. 1736), and do not depend on the number of **DDSDataWriter** (p. 1589) instances the **DDSFlowController** (p. 1736) is servicing. For instance, the specified flow rate does *not* double simply because two **DDSDataWriter** (p. 1589) instances are waiting to write.

Entity:

DDSFlowController (p. 1736)

Properties:

RxO (p. 556) = N/A

Changeable (p. 556) = **NO** (p. 556) for **DDS_FlowControllerProperty_t::scheduling_policy** (p. 1157), **YES** (p. 556) for **DDS_FlowControllerProperty_t::token_bucket** (p. 1157). However, the special value of **DDS_DURATION_INFINITE** (p. 522) as **DDS_FlowControllerTokenBucketProperty_t::period** (p. 1159) is strictly used to create an *on-demand* **DDSFlowController** (p. 1736). The token period cannot toggle from an infinite to finite value (or vice versa). It can, however, change from one finite value to another.

9.79.2 Member Data Documentation

9.79.2.1 scheduling_policy

DDS_FlowControllerSchedulingPolicy DDS_FlowControllerProperty_t::scheduling_policy

Scheduling policy.

Determines the scheduling policy for servicing the **DDSDataWriter** (p. 1589) instances associated with the **DDSFlowController** (p. 1736).

[default] DDS_EDF_FLOW_CONTROLLER_SCHED_POLICY (p. 124)

9.79.2.2 token_bucket

struct **DDS_FlowControllerTokenBucketProperty_t** DDS_FlowControllerProperty_t::token_bucket

Settings for the token bucket.

9.80 DDS_FlowControllerTokenBucketProperty_t Struct Reference

DDSFlowController (p. 1736) uses the popular token bucket approach for open loop network flow control. The flow control characteristics are determined by the token bucket properties.

Public Attributes

- **DDS_Long max_tokens**

  Maximum number of tokens than can accumulate in the token bucket.
- **DDS_Long tokens_added_per_period**

  The number of tokens added to the token bucket per specified period.
- **DDS_Long tokens_leaked_per_period**

  The number of tokens removed from the token bucket per specified period.
- struct **DDS_Duration_t period**

  Period for adding tokens to and removing tokens from the bucket.
- **DDS_Long bytes_per_token**

  Maximum number of bytes allowed to send for each token available.

9.80.1 Detailed Description

DDSFlowController (p. 1736) uses the popular token bucket approach for open loop network flow control. The flow control characteristics are determined by the token bucket properties.

Asynchronously published samples are queued up and transmitted based on the token bucket flow control scheme. The token bucket contains tokens, each of which represents a number of bytes. Samples can be sent only when there are sufficient tokens in the bucket. As samples are sent, tokens are consumed. The number of tokens consumed is proportional to the size of the data being sent. Tokens are replenished on a periodic basis.

The rate at which tokens become available and other token bucket properties determine the network traffic flow.

Note that if the same sample must be sent to multiple destinations, separate tokens are required for each destination. Only when multiple samples are destined to the same destination will they be co-alesced and sent using the same token(s). In other words, each token can only contribute to a single network packet.

Entity:

DDSFlowController (p. 1736)

Properties:

RxO (p. 556) = N/A

Changeable (p. 556) = **YES** (p. 556). However, the special value of **DDS_DURATION_INFINITE** (p. 522) as **DDS_FlowControllerTokenBucketProperty_t::period** (p. 1159) is strictly used to create an *on-demand* **DDSFlowController** (p. 1736). The token period cannot toggle from an infinite to finite value (or vice versa). It can, however, change from one finite value to another.

9.80.2 Member Data Documentation

9.80.2.1 max_tokens

```
DDS_Long DDS_FlowControllerTokenBucketProperty_t::max_tokens
```

Maximum number of tokens than can accumulate in the token bucket.

The number of tokens in the bucket will never exceed this value. Any excess tokens are discarded. This property value, combined with **DDS_FlowControllerTokenBucketProperty_t::bytes_per_token** (p. 1159), determines the maximum allowable data burst.

Use **DDS_LENGTH_UNLIMITED** (p. 636) to allow accumulation of an unlimited amount of tokens (and therefore potentially an unlimited burst size).

[default] DDS_LENGTH_UNLIMITED (p. 636)

[range] [1,**DDS_LENGTH_UNLIMITED** (p. 636)]

9.80.2.2 tokens_added_per_period

DDS_Long DDS_FlowControllerTokenBucketProperty_t::tokens_added_per_period

The number of tokens added to the token bucket per specified period.

DDSFlowController (p. 1736) transmits data only when tokens are available. Tokens are periodically replenished. This field determines the number of tokens added to the token bucket with each periodic replenishment.

Available tokens are distributed to associated **DDSDataWriter** (p. 1589) instances based on the **DDS_FlowController↩ Property_t::scheduling_policy** (p. 1157).

Use **DDS_LENGTH_UNLIMITED** (p. 636) to add the maximum number of tokens allowed by **DDS_FlowController↩ TokenBucketProperty_t::max_tokens** (p. 1158).

[default] DDS_LENGTH_UNLIMITED (p. 636)

[range] [1,**DDS_LENGTH_UNLIMITED** (p. 636)]

9.80.2.3 tokens_leaked_per_period

DDS_Long DDS_FlowControllerTokenBucketProperty_t::tokens_leaked_per_period

The number of tokens removed from the token bucket per specified period.

DDSFlowController (p. 1736) transmits data only when tokens are available. When tokens are replenished and there are sufficient tokens to send all samples in the queue, this property determines whether any or all of the leftover tokens remain in the bucket.

Use **DDS_LENGTH_UNLIMITED** (p. 636) to remove all excess tokens from the token bucket once all samples have been sent. In other words, no token accumulation is allowed. When new samples are written after tokens were purged, the earliest point in time at which they can be sent is at the next periodic replenishment.

[default] 0

[range] [0,**DDS_LENGTH_UNLIMITED** (p. 636)]

9.80.2.4 period

struct **DDS_Duration_t** DDS_FlowControllerTokenBucketProperty_t::period

Period for adding tokens to and removing tokens from the bucket.

DDSFlowController (p. 1736) transmits data only when tokens are available. This field determines the period by which tokens are added or removed from the token bucket.

The special value **DDS_DURATION_INFINITE** (p. 522) can be used to create an *on-demand* **DDSFlowController** (p. 1736), for which tokens are no longer replenished periodically. Instead, tokens must be added explicitly by calling **DDSFlowController::trigger_flow** (p. 1738). This external trigger adds **DDS_FlowControllerTokenBucket↩ Property_t::tokens_added_per_period** (p. 1158) tokens each time it is called (subject to the other property settings).

[default] 1 second

[range] [1 nanosec, 1 year] or **DDS_DURATION_INFINITE** (p. 522)

9.80.2.5 bytes_per_token

```
DDS_Long DDS_FlowControllerTokenBucketProperty_t::bytes_per_token
```

Maximum number of bytes allowed to send for each token available.

DDSFlowController (p. 1736) transmits data only when tokens are available. This field determines the number of bytes that can actually be transmitted based on the number of tokens.

Tokens are always consumed in whole by each **DDSDataWriter** (p. 1589). That is, in cases where **DDS_Flow↩ ControllerTokenBucketProperty_t::bytes_per_token** (p. 1159) is greater than the sample size, multiple samples may be sent to the same destination using a single token (regardless of **DDS_FlowControllerProperty_t::scheduling_↩ policy** (p. 1157)).

Where fragmentation is required, the fragment size will be **DDS_FlowControllerTokenBucketProperty_t::bytes↩ _per_token** (p. 1159) or the minimum largest message size across all transports installed with the **DDSDataWriter** (p. 1589), whichever is less.

Use **DDS_LENGTH_UNLIMITED** (p. 636) to indicate that an unlimited number of bytes can be transmitted per token. In other words, a single token allows the recipient **DDSDataWriter** (p. 1589) to transmit all its queued samples to a single destination. A separate token is required to send to each additional destination.

[default] DDS_LENGTH_UNLIMITED (p. 636)

[range] [1024,**DDS_LENGTH_UNLIMITED** (p. 636)]

9.81 DDS_GroupDataQosPolicy Struct Reference

Attaches a buffer of opaque data that is distributed by means of **Built-in Topics** (p. 61) during discovery.

Public Attributes

- struct **DDS_OctetSeq value**
  *<<**spec**>> (p. 209) a sequence of octets*

9.81.1 Detailed Description

Attaches a buffer of opaque data that is distributed by means of **Built-in Topics** (p. 61) during discovery.

Entity:

DDSPublisher (p. 1818), **DDSSubscriber** (p. 1860)

Properties:

RxO (p. 556) = NO
Changeable (p. 556) = **YES** (p. 556)

See also

DDSDomainParticipant::get_builtin_subscriber (p. 1661)

9.81.2 Usage

The additional information is attached to a **DDSPublisher** (p. 1818) or **DDSSubscriber** (p. 1860). This extra data is not used by RTI Connext itself. When a remote application discovers the **DDSPublisher** (p. 1818) or **DDSSubscriber** (p. 1860), it can access that information and use it for its own purposes.

Use cases for this QoS policy, as well as the **DDS_TopicDataQosPolicy** (p. 1396) and **DDS_UserDataQosPolicy** (p. 1504), are often application-to-application identification, authentication, authorization, and encryption purposes. For example, applications can use Group or User Data to send security certificates to each other for RSA-type security.

In combination with **DDSDataReaderListener** (p. 1583), **DDSDataWriterListener** (p. 1612) and operations such as **DDSDomainParticipant::ignore_publication** (p. 1664) and **DDSDomainParticipant::ignore_subscription** (p. 1664), this QoS policy can help an application to define and enforce its own security policies. For example, an application can implement matching policies similar to those of the **DDS_PartitionQosPolicy** (p. 1240), except the decision can be made based on an application-defined policy.

The use of this QoS is not limited to security; it offers a simple, yet flexible extensibility mechanism.

Important: RTI Connext stores the data placed in this policy in pre-allocated pools. It is therefore necessary to configure RTI Connext with the maximum size of the data that will be stored in policies of this type. This size is configured with **DDS_DomainParticipantResourceLimitsQosPolicy::publisher_group_data_max_length** (p. 992) and **DDS_DomainParticipantResourceLimitsQosPolicy::subscriber_group_data_max_length** (p. 992).

9.81.3 Member Data Documentation

9.81.3.1 value

`struct` **DDS_OctetSeq** DDS_GroupDataQosPolicy::value

<<***spec***>> *(p. 209)* a sequence of octets

[default] Empty (zero-sized)

[range] Octet sequence of length [0,max_length]

Categories

Mutable (p. 369), **Application-Defined Metadata** (p. 254), **Discovery Metadata** (p. 377)

9.82 DDS_GUID_t Struct Reference

Type for *GUID* (Global Unique Identifier) representation.

Public Attributes

- **DDS_Octet value** [DDS_GUID_LENGTH]

  A 16 byte array containing the GUID value.

9.82.1 Detailed Description

Type for *GUID* (Global Unique Identifier) representation.

Represents a 128 bit GUID.

Alternative representation of **DDS_RTPS_GUID_t** (p. 1314). Memory and wire representation for this type is the same as the one for **DDS_RTPS_GUID_t** (p. 1314).

9.82.2 Member Data Documentation

9.82.2.1 value

```
DDS_Octet DDS_GUID_t::value[DDS_GUID_LENGTH]
```

A 16 byte array containing the GUID value.

9.83 DDS_HistoryQosPolicy Struct Reference

<<***spec***>> *(p. 209)* Specifies the behavior of RTI Connext in the case where the value of a sample changes (one or more times) before it can be successfully communicated to one or more existing subscribers.

Public Attributes

- **DDS_HistoryQosPolicyKind kind**

  <<***basic***>> *(p. 201)* <<***spec***>> *(p. 209) Specifies the kind of history to be kept.*

- **DDS_Long depth**

  <<***basic***>> *(p. 201)* <<***spec***>> *(p. 209) Specifies the number of samples per instance to be kept, when the* `kind` *is* ***DDS_KEEP_LAST_HISTORY_QOS*** *(p. 604)*

9.83.1 Detailed Description

<<***spec***>> *(p. 209)* Specifies the behavior of RTI Connext in the case where the value of a sample changes (one or more times) before it can be successfully communicated to one or more existing subscribers.

This QoS policy specifies how much data must to stored by RTI Connext for a **DDSDataWriter** (p. 1589) or **DDSDataReader** (p. 1556). It controls whether RTI Connext should deliver only the most recent value, attempt to deliver all intermediate values, or do something in between.

On the publishing side, this QoS policy controls the samples that should be maintained by the **DDSDataWriter** (p. 1589) on behalf of existing **DDSDataReader** (p. 1556) entities. The behavior with regards to a **DDSDataReader** (p. 1556) entities discovered after a sample is written is controlled by the **DURABILITY** (p. 596) policy.

On the subscribing side, this QoS policy controls the samples that should be maintained until the application "takes" them from RTI Connext.

Entity:

DDSTopic (p. 1885), **DDSDataReader** (p. 1556), **DDSDataWriter** (p. 1589)

Properties:

RxO (p. 556) = NO
Changeable (p. 556) = **UNTIL ENABLE** (p. 556)

See also

DDS_ReliabilityQosPolicy (p. 1295)

DDS_HistoryQosPolicy (p. 1162)

9.83.2 Usage

This policy controls the behavior of RTI Connext when the value of an instance changes before it is finally communicated to **DDSDataReader** (p. 1556) entities.

When a **DDSDataWriter** (p. 1589) sends data, or a **DDSDataReader** (p. 1556) receives data, the data sent or received is stored in a cache whose contents are controlled by this QoS policy. This QoS policy interacts with **DDS_ReliabilityQosPolicy** (p. 1295) by controlling whether RTI Connext guarantees that *all* of the sent data is received (**DDS_KEEP_ALL_HISTORY_QOS** (p. 604)) or if only the last N data values sent are guaranteed to be received (**DDS_KEEP_LAST_HISTORY_QOS** (p. 604))–this is a reduced level of reliability.

The amount of data that is sent to new DataReaders who have configured their **DDS_DurabilityQosPolicy** (p. 1005) to receive previously published data is controlled by **DDS_DurabilityQosPolicy::writer_depth** (p. 1008), not by the History QoS policy.

Note that the History QoS policy does not control the *physical* sizes of the send and receive queues. The memory allocation for the queues is controlled by the **DDS_ResourceLimitsQosPolicy** (p. 1308).

If kind is **DDS_KEEP_LAST_HISTORY_QOS** (p. 604) (the default), then RTI Connext will only attempt to keep the latest values of the instance and discard the older ones. In this case, the value of depth regulates the maximum number of values (up to and including the most current one) RTI Connext will maintain and deliver until the samples are fully acknowledged. After N values have been sent or received, any new data will overwrite the oldest data in the queue. Thus the queue acts like a circular buffer of length *N*.

For keyed-data, there is different behavior on the publishing and subscribing sides associated with how invalid samples representing the disposal of or unregistration from an instance affect history.

On the publishing side, unregistering from or disposing of an instance creates an invalid sample that is accounted for in the history depth. This means that an invalid sample may replace a value that is currently being stored in the writer queue.

On the subscribing side, however, invalid samples do not count towards history depth and will not replace a value that is being stored in the reader queue.

On both the publishing and subscribing sides, there can only ever be one invalid sample per-instance and that one sample can be in different states depending on whether the instance has been disposed, unregistered, or both.

The default (and most common setting) for depth is 1, indicating that only the most recent value should be delivered.

If kind is **DDS_KEEP_ALL_HISTORY_QOS** (p. 604), then RTI Connext will attempt to maintain and deliver all the values of the instance to existing subscribers. The resources that RTI Connext can use to keep this history are limited by the settings of the **RESOURCE_LIMITS** (p. 635). If the limit is reached, then the behavior of RTI Connext will depend on the **RELIABILITY** (p. 631). If the Reliability kind is **DDS_BEST_EFFORT_RELIABILITY_QOS** (p. 633), then the old values will be discarded. If Reliability kind is RELIABLE, then RTI Connext will block the **DDSDataWriter** (p. 1589) until it can deliver the necessary old values to all subscribers.

9.83.3 Consistency

This QoS policy's depth must be consistent with the **RESOURCE_LIMITS** (p. 635) max_samples_per_↩ instance. For these two QoS to be consistent, they must verify that *depth <= max_samples_per_instance*.

See also

DDS_ResourceLimitsQosPolicy (p. 1308)

9.83.4 Member Data Documentation

9.83.4.1 kind

DDS_HistoryQosPolicyKind DDS_HistoryQosPolicy::kind

*<<**basic**>> (p. 201) <<**spec**>> (p. 209)* Specifies the kind of history to be kept.

For DataWriters, the samples are only kept either until they are fully acknowledged by all matching DataReaders or until they are replaced or removed. Samples can be replaced or removed for a number of reasons, including but not limited to **DDS_LifespanQosPolicy** (p. 1175) expiration, replacement because the **DDS_HistoryQosPolicy::depth** (p. 1164) has been exceeded, or one of the **DDS_ResourceLimitsQosPolicy** (p. 1308) settings has been exceeded.

[default] DDS_KEEP_LAST_HISTORY_QOS (p. 604)

Categories

Immutable (p. 333), **Applied Per-Instance** (p. 375), **Reliability** (p. 314), **Data Availability** (p. 261)

9.83.4.2 depth

DDS_Long DDS_HistoryQosPolicy::depth

<<***basic***>> *(*p. *201)* <<***spec***>> *(*p. *209)* Specifies the number of samples per instance to be kept, when the `kind` is **DDS_KEEP_LAST_HISTORY_QOS** (p. 604)

If a value other than 1 (the default) is specified, it should be consistent with the settings of the **RESOURCE_LIMITS** (p. 635) policy. That is:

depth <= **DDS_ResourceLimitsQosPolicy::max_samples_per_instance** (p. 1311)

When the `kind` is **DDS_KEEP_ALL_HISTORY_QOS** (p. 604), the depth has no effect. Its implied value is `infinity` (in practice limited by the settings of the **RESOURCE_LIMITS** (p. 635) policy).

When a DataWriter responds to a TopicQuery, the samples that are evaluated against the TopicQuery filter are only those samples that fall within the **DDS_DurabilityQosPolicy::writer_depth** (p. 1008), not this depth.

[default] 1

[range] [1,100 million], <= **DDS_ResourceLimitsQosPolicy::max_samples_per_instance** (p. 1311)

Categories

Immutable (p. 333), **Applied Per-Instance** (p. 375), **Reliability** (p. 314), **Data Availability** (p. 261)

9.84 DDS_InconsistentTopicStatus Struct Reference

DDS_INCONSISTENT_TOPIC_STATUS (p. 539)

Public Attributes

- **DDS_Long total_count**

  *Total cumulative count of the pairs (**DDSDataReader** (p. 1556),**DDSDataWriter** (p. 1589)) whose topic names match the **DDSTopic** (p. 1885) to which this status is attached and whose types are inconsistent according to the rules defined in **DDS_TypeConsistencyEnforcementQosPolicy** (p. 1493).*

- **DDS_Long total_count_change**

  *The incremental number of inconsistent pairs (**DDSDataReader** (p. 1556),**DDSDataWriter** (p. 1589)) for the **DDSTopic** (p. 1885) to which this status is attached, that have been discovered since the last time this status was read.*

9.84.1 Detailed Description

DDS_INCONSISTENT_TOPIC_STATUS (p. 539)

Entity:

DDSTopic (p. 1885)

Listener:

DDSTopicListener (p. 1894)

Every time a **DDSDataReader** (p. 1556) and **DDSDataWriter** (p. 1589) with the same **DDSTopic** (p. 1885) do not match because the type-consistency enforcement policy fails, the inconsistent topic status is updated for the **DDSTopic** (p. 1885).

See also

DDS_TypeConsistencyEnforcementQosPolicy (p. 1493)

9.84.2 Member Data Documentation

9.84.2.1 total_count

```
DDS_Long DDS_InconsistentTopicStatus::total_count
```

Total cumulative count of the pairs (**DDSDataReader** (p. 1556),**DDSDataWriter** (p. 1589)) whose topic names match the **DDSTopic** (p. 1885) to which this status is attached and whose types are inconsistent according to the rules defined in **DDS_TypeConsistencyEnforcementQosPolicy** (p. 1493).

9.84.2.2 total_count_change

```
DDS_Long DDS_InconsistentTopicStatus::total_count_change
```

The incremental number of inconsistent pairs (**DDSDataReader** (p. 1556),**DDSDataWriter** (p. 1589)) for the **DDSTopic** (p. 1885) to which this status is attached, that have been discovered since the last time this status was read.

9.85 DDS_InstanceHandleSeq Struct Reference

Instantiates **FooSeq** (p. 1965) < **DDS_InstanceHandle_t** (p. 76) > .

9.85.1 Detailed Description

Instantiates **FooSeq** (p. 1965) < **DDS_InstanceHandle_t** (p. 76) > .

Instantiates:

<<***generic***>> *(*p. *433)* **FooSeq** (p. 1965)

See also

DDS_InstanceHandle_t (p. 76)

FooSeq (p. 1965)

9.86 DDS_Int8Seq Struct Reference

Instantiates **FooSeq** (p. 1965) < **DDS_Int8** (p. 514) >

9.86.1 Detailed Description

Instantiates **FooSeq** (p. 1965) < **DDS_Int8** (p. 514) >

Instantiates:

<<***generic***>> *(*p. *433)* **FooSeq** (p. 1965)

See also

DDS_Int8 (p. 514)

FooSeq (p. 1965)

9.87 DDS_InvalidLocalAdvanceNoticeStatus Struct Reference

DDS_INVALID_LOCAL_ADVANCE_NOTICE_STATUS (p. 543)

Public Attributes

- struct **DDS_Time_t expiration_time**

  The time when the local artifact will expire.
- **DDS_UnsignedLong expiration_code**

  The reason (a numeric code) for the local artifact to expire.
- const char * **expiration_message**

  A descriptive message for the ***DDS_InvalidLocalAdvanceNoticeStatus::expiration_code*** *(*p. *1168) code.*

9.87.1 Detailed Description

DDS_INVALID_LOCAL_ADVANCE_NOTICE_STATUS (p. 543)

Currently, this status is only triggered when enabling the RTI Security Plugins. Please refer to the `RTI Security Plugins User's Manual` for more information.

9.87.2 Member Data Documentation

9.87.2.1 expiration_time

`struct` **`DDS_Time_t`** `DDS_InvalidLocalAdvanceNoticeStatus::expiration_time`

The time when the local artifact will expire.

9.87.2.2 expiration_code

`DDS_UnsignedLong` `DDS_InvalidLocalAdvanceNoticeStatus::expiration_code`

The reason (a numeric code) for the local artifact to expire.

9.87.2.3 expiration_message

`const char* DDS_InvalidLocalAdvanceNoticeStatus::expiration_message`

A descriptive message for the **DDS_InvalidLocalAdvanceNoticeStatus::expiration_code** (p. 1168) code.

9.88 DDS_KeyedOctets Struct Reference

Built-in type consisting of a variable-length array of opaque bytes and a string that is the key.

Public Member Functions

- **DDS_KeyedOctets** ()
  Constructor.
- **DDS_KeyedOctets** (int key_size, int size)
  Constructor that specifies the allocated sizes.
- ~**DDS_KeyedOctets** ()
  Destructor.

Public Attributes

- char * **key**

  Instance key associated with the specified value.
- int **length**

  Number of octets to serialize.
- unsigned char * **value**

  DDS_Octets *(p. 1217) array value.*

9.88.1 Detailed Description

Built-in type consisting of a variable-length array of opaque bytes and a string that is the key.

9.88.2 Constructor & Destructor Documentation

9.88.2.1 DDS_KeyedOctets() [1/2]

```
DDS_KeyedOctets::DDS_KeyedOctets ( )  [inline]
```

Constructor.

The default constructor initializes the newly created object with NULL key, NULL value, and zero length.

9.88.2.2 DDS_KeyedOctets() [2/2]

```
DDS_KeyedOctets::DDS_KeyedOctets (
            int key_size,
            int size )  [inline]
```

Constructor that specifies the allocated sizes.

After this method is called, key is initialized with the empty string and length is set to zero.

If a memory allocation failure occurs, and the **DDS_KeyedOctets** (p. 1168) structure is allocated but the array and/or key string inside of it cannot be, the unallocated member will be NULL.

Parameters

key_size	<<***in***>> *(p. 433)* Size of the allocated key (with NULL-terminated character). Cannot be smaller than one.
size	<<***in***>> *(p. 433)* Size of the allocated octets array.

References **DDS_OctetBuffer_alloc()**, **DDS_String_alloc()**, **DDS_String_free()**, **key**, and **value**.

9.88.2.3 ~DDS_KeyedOctets()

```
DDS_KeyedOctets::~DDS_KeyedOctets ( )  [inline]
```

Destructor.

References **DDS_OctetBuffer_free()**, **DDS_String_free()**, **key**, and **value**.

9.88.3 Member Data Documentation

9.88.3.1 key

```
char* DDS_KeyedOctets::key
```

Instance key associated with the specified value.

Referenced by **DDS_KeyedOctets()**, and ~**DDS_KeyedOctets()**.

9.88.3.2 length

```
int DDS_KeyedOctets::length
```

Number of octets to serialize.

9.88.3.3 value

```
unsigned char* DDS_KeyedOctets::value
```

DDS_Octets (p. 1217) array value.

Referenced by **DDS_KeyedOctets()**, and ~**DDS_KeyedOctets()**.

9.89 DDS_KeyedOctetsSeq Struct Reference

Instantiates **FooSeq** (p. 1965) < **DDS_KeyedOctets** (p. 1168) >.

9.89.1 Detailed Description

Instantiates **FooSeq** (p. 1965) < **DDS_KeyedOctets** (p. 1168) >.

Instantiates:

<<*generic*>> *(*p. *433)* **FooSeq** (p. 1965)

See also

DDS_KeyedOctets (p. 1168)

9.90 DDS_KeyedString Struct Reference

Keyed string built-in type.

Public Member Functions

- **DDS_KeyedString** ()
  Constructor.
- **DDS_KeyedString** (int key_size, int size)
  Constructor that specifies the allocated sizes.
- ~**DDS_KeyedString** ()
  Destructor.

Public Attributes

- char * **key**
  Instance key associated with the specified value.
- char * **value**
  String value.

9.90.1 Detailed Description

Keyed string built-in type.

9.90.2 Constructor & Destructor Documentation

9.90.2.1 DDS_KeyedString() [1/2]

```
DDS_KeyedString::DDS_KeyedString ( )  [inline]
```

Constructor.

The default constructor initializes the newly created object with NULL key and value.

9.90.2.2 DDS_KeyedString() [2/2]

```
DDS_KeyedString::DDS_KeyedString (
            int key_size,
            int size )  [inline]
```

Constructor that specifies the allocated sizes.

The allocated strings are initialized to empty ("").

If a memory allocation failure occurs, and the **DDS_KeyedString** (p. 1171) structure is allocated but one or both strings inside of it cannot be, the unallocated string will be NULL.

Parameters

key_size	<<***in***>> *(p. 433)* Size of the allocated key string (with NULL-terminated character). Cannot be smaller than one.
size	<<***in***>> *(p. 433)* Size of the allocated value string (with NULL-terminated character). Cannot be smaller than one.

References **DDS_String_alloc()**, **DDS_String_free()**, **key**, and **value**.

9.90.2.3 ~DDS_KeyedString()

```
DDS_KeyedString::~DDS_KeyedString ( )  [inline]
```

Destructor.

References **DDS_String_free()**, **key**, and **value**.

9.90.3 Member Data Documentation

9.90.3.1 key

```
char* DDS_KeyedString::key
```

Instance key associated with the specified value.

Referenced by **DDS_KeyedString()**, and ~**DDS_KeyedString()**.

9.90.3.2 value

```
char* DDS_KeyedString::value
```

String value.

Referenced by **DDS_KeyedString()**, and ~**DDS_KeyedString()**.

9.91 DDS_KeyedStringSeq Struct Reference

Instantiates **FooSeq** (p. 1965) < **DDS_KeyedString** (p. 1171) > .

9.91.1 Detailed Description

Instantiates **FooSeq** (p. 1965) < **DDS_KeyedString** (p. 1171) > .

Instantiates:

<<***generic***>> *(*p. *433)* **FooSeq** (p. 1965)

See also

DDS_KeyedString (p. 1171)

9.92 DDS_LatencyBudgetQosPolicy Struct Reference

<<***spec***>> *(*p. *209)* Provides a hint as to the maximum acceptable delay from the time the data is written to the time it is received by the subscribing applications.

Public Attributes

- struct **DDS_Duration_t duration**

  <<***spec***>> *(*p. *209) Duration of the maximum acceptable delay.*

9.92.1 Detailed Description

<<***spec***>> *(p. 209)* Provides a hint as to the maximum acceptable delay from the time the data is written to the time it is received by the subscribing applications.

This policy is a *hint* to a DDS implementation; it can be used to change how it processes and sends data that has low latency requirements. The DDS specification does not mandate whether or how this policy is used.

Entity:

DDSTopic (p. 1885), **DDSDataReader** (p. 1556), **DDSDataWriter** (p. 1589)

Status:

DDS_OFFERED_INCOMPATIBLE_QOS_STATUS (p. 540), **DDS_REQUESTED_INCOMPATIBLE_QOS_↩ STATUS** (p. 540)

Properties:

RxO (p. 556) = YES
Changeable (p. 556) = **YES** (p. 556)

See also

DDS_PublishModeQosPolicy (p. 1279)

DDSFlowController (p. 1736)

9.92.2 Usage

This policy provides a means for the application to indicate to the middleware the urgency of the data communication. By having a non-zero `duration`, RTI Connext can optimize its internal operation.

RTI Connext uses it in conjunction with **DDS_ASYNCHRONOUS_PUBLISH_MODE_QOS** (p. 629) **DDSDataWriter** (p. 1589) instances associated with a **DDS_EDF_FLOW_CONTROLLER_SCHED_POLICY** (p. 124) **DDSFlow↩ Controller** (p. 1736) only. Together with the time of write, **DDS_LatencyBudgetQosPolicy::duration** (p. 1175) determines the deadline of each individual sample. RTI Connext uses this information to prioritize the sending of asynchronously published data; see **DDS_AsynchronousPublisherQosPolicy** (p. 798).

9.92.3 Compatibility

The value offered is considered compatible with the value requested if and only if the inequality *offered duration* <= *requested duration* evaluates to 'TRUE'.

9.92.4 Member Data Documentation

9.92.4.1 duration

`struct DDS_Duration_t DDS_LatencyBudgetQosPolicy::duration`

<<***spec***>> *(p. 209)* Duration of the maximum acceptable delay.

[default] 0 (meaning minimize the delay)

Categories

Mutable (p. 369), **Discovery Metadata** (p. 377), **Matching** (p. 382), **Data Availability** (p. 261)

9.93 DDS_LifespanQosPolicy Struct Reference

<<***spec***>> *(p. 209)* Specifies how long the data written by the **DDSDataWriter** (p. 1589) is considered valid.

Public Attributes

- struct **DDS_Duration_t duration**
  <<***basic***>> (p. *201)* <<***spec***>> (p. *209) Maximum duration for the data's validity.*

9.93.1 Detailed Description

<<***spec***>> *(p. 209)* Specifies how long the data written by the **DDSDataWriter** (p. 1589) is considered valid.

Each data sample written by the **DDSDataWriter** (p. 1589) has an associated expiration time beyond which the data should not be delivered to any application. Once the sample expires, the data will be removed from the **DDSDataReader** (p. 1556) caches as well as from the transient and persistent information caches.

The expiration time of each sample from the **DDSDataWriter** (p. 1589)'s cache is computed by adding the duration specified by this QoS policy to the time when the sample is added to the **DDSDataWriter** (p. 1589)'s cache. This timestamp is not necessarily equal to the sample's source timestamp that can be provided by the user using the **Foo↩ DataWriter::write_w_timestamp** (p. 1955) or **FooDataWriter::write_w_params** (p. 1956) API.

The expiration time of each sample from the **DDSDataReader** (p. 1556)'s cache is computed by adding the duration to the reception timestamp.

See also

FooDataWriter::write (p. 1951)

FooDataWriter::write_w_timestamp (p. 1955)

Entity:

DDSTopic (p. 1885), **DDSDataWriter** (p. 1589)

Properties:

RxO (p. 556) = N/A
Changeable (p. 556) = **YES** (p. 556)

9.93.2 Usage

The Lifespan QoS policy can be used to control how much data is stored by RTI Connext. Even if it is configured to store "all" of the data sent or received for a topic (see **DDS_HistoryQosPolicy** (p. 1162)), the total amount of data it stores may be limited by this QoS policy.

You may also use this QoS policy to ensure that applications do not receive or act on data, commands or messages that are too old and have 'expired.'

To avoid inconsistencies, multiple writers of the same instance should have the same lifespan.

See also

DDS_SampleInfo::source_timestamp (p. 1348)

DDS_SampleInfo::reception_timestamp (p. 1351)

9.93.3 Member Data Documentation

9.93.3.1 duration

`struct` **`DDS_Duration_t`** `DDS_LifespanQosPolicy::duration`

*<<**basic**>> (p. 201) <<**spec**>> (p. 209)* Maximum duration for the data's validity.

[default] DDS_DURATION_INFINITE (p. 522)

[range] [1 nanosec, 1 year] or **DDS_DURATION_INFINITE** (p. 522)

Categories

Mutable (p. 369), **Discovery Metadata** (p. 377), **Durability** (p. 278), **Data Availability** (p. 261)

9.94 DDS_LivelinessChangedStatus Struct Reference

DDS_LIVELINESS_CHANGED_STATUS (p. 542)

Public Attributes

- **DDS_Long alive_count**

  The total count of currently alive ***DDSDataWriter*** *(*p. *1589) entities that write the* ***DDSTopic*** *(*p. *1885) that this* ***DDSDataReader*** *(*p. *1556) reads.*
- **DDS_Long not_alive_count**

  The total count of currently not_alive ***DDSDataWriter*** *(*p. *1589) entities that write the* ***DDSTopic*** *(*p. *1885) that this* ***DDSDataReader*** *(*p. *1556) reads.*
- **DDS_Long alive_count_change**

  The change in the alive_count since the last time the listener was called or the status was read.
- **DDS_Long not_alive_count_change**

  The change in the not_alive_count since the last time the listener was called or the status was read.
- **DDS_InstanceHandle_t last_publication_handle**

  This InstanceHandle can be used to look up which remote ***DDSDataWriter*** *(*p. *1589) was the last to cause this DataReader's status to change, using* ***DDSDataReader::get_matched_publication_data*** *(*p. *1568).*

9.94.1 Detailed Description

DDS_LIVELINESS_CHANGED_STATUS (p. 542)

The **DDSDataReaderListener::on_liveliness_changed** (p. 1584) callback may be invoked for the following reasons:

- The liveliness of any **DDSDataWriter** (p. 1589) matching this DataReader (as defined by the **DDS_LivelinessQosPolicyKind** (p. 607) setting) is lost.
- A DataWriter's liveliness is recovered after being lost.
- A new matching DataWriter has been discovered.
- A matching DataWriter has been deleted.
- A QoS Policy has changed such that a DataWriter that matched this DataReader before no longer matches (such as a change to the **DDS_PartitionQosPolicy** (p. 1240)). In this case, RTI Connext will no longer keep track of the DataWriter's liveliness. Furthermore, consider two scenarios:
  - DataWriter was alive when it and DataReader stopped matching: **DDS_LivelinessChangedStatus::alive_count** (p. 1178) will decrease (since there's one less matching alive DataWriter) and **DDS_LivelinessChangedStatus::not_alive_count** (p. 1178) will remain the same (since the DataWriter is still alive).
  - DataWriter was not alive when it and DataReader stopped matching: **DDS_LivelinessChangedStatus::alive_count** (p. 1178) will remain the same (since the matching DataWriter was not alive) and **DDS_LivelinessChangedStatus::not_alive_count** (p. 1178) will decrease (since there's one less not-alive matching DataWriter).

    Note: There are several ways that a DataWriter and DataReader can become incompatible after the DataWriter has lost liveliness. For example, when the **DDS_LivelinessQosPolicyKind** (p. 607) is set to DDS_MANUAL_BY_PARTICIPANT_LIVELINESS_QOS, it is possible that the DataWriter has not asserted its liveliness in a timely manner, and then a QoS change occurs on the DataWriter or DataReader that makes the entities incompatible.
- A QoS Policy (such as the **DDS_PartitionQosPolicy** (p. 1240)) has changed such that a DataWriter that was unmatched with the DataReader now matches.

9.94.2 Member Data Documentation

9.94.2.1 alive_count

DDS_Long DDS_LivelinessChangedStatus::alive_count

The total count of currently alive **DDSDataWriter** (p. 1589) entities that write the **DDSTopic** (p. 1885) that this **DDSDataReader** (p. 1556) reads.

9.94.2.2 not_alive_count

DDS_Long DDS_LivelinessChangedStatus::not_alive_count

The total count of currently not_alive **DDSDataWriter** (p. 1589) entities that write the **DDSTopic** (p. 1885) that this **DDSDataReader** (p. 1556) reads.

9.94.2.3 alive_count_change

DDS_Long DDS_LivelinessChangedStatus::alive_count_change

The change in the alive_count since the last time the listener was called or the status was read.

9.94.2.4 not_alive_count_change

DDS_Long DDS_LivelinessChangedStatus::not_alive_count_change

The change in the not_alive_count since the last time the listener was called or the status was read.

Note that a positive not_alive_count_change means one of the following:

- The DomainParticipant containing the matched DataWriter has lost liveliness or has been deleted.
- The matched DataWriter has lost liveliness or has been deleted.

9.94.2.5 last_publication_handle

```
DDS_InstanceHandle_t DDS_LivelinessChangedStatus::last_publication_handle
```

This InstanceHandle can be used to look up which remote **DDSDataWriter** (p. 1589) was the last to cause this Data↩ Reader's status to change, using **DDSDataReader::get_matched_publication_data** (p. 1568).

It's possible that the DataWriter has been purged from the discovery database. (See the "Discovery Overview" section of the User's Manual.) If so, the **DDSDataReader::get_matched_publication_data** (p. 1568) method will not be able to return information about the DataWriter. In this case, the only way to get information about the lost DataWriter is if you cached the information previously.

9.95 DDS_LivelinessLostStatus Struct Reference

DDS_LIVELINESS_LOST_STATUS (p. 542)

Public Attributes

- **DDS_Long total_count**

  *Total cumulative number of times that a previously-alive **DDSDataWriter** (p. 1589) became not alive due to a failure to to actively signal its liveliness within the offered liveliness period.*
- **DDS_Long total_count_change**

  The incremental changes in total_count since the last time the listener was called or the status was read.

9.95.1 Detailed Description

DDS_LIVELINESS_LOST_STATUS (p. 542)

Entity:

DDSDataWriter (p. 1589)

Listener:

DDSDataWriterListener (p. 1612)

The liveliness that the **DDSDataWriter** (p. 1589) has committed through its **DDS_LivelinessQosPolicy** (p. 1180) was not respected; thus **DDSDataReader** (p. 1556) entities will consider the **DDSDataWriter** (p. 1589) as no longer "alive/active".

9.95.2 Member Data Documentation

9.95.2.1 total_count

```
DDS_Long DDS_LivelinessLostStatus::total_count
```

Total cumulative number of times that a previously-alive **DDSDataWriter** (p. 1589) became not alive due to a failure to to actively signal its liveliness within the offered liveliness period.

This count does not change when an already not alive **DDSDataWriter** (p. 1589) simply remains not alive for another liveliness period.

9.95.2.2 total_count_change

```
DDS_Long DDS_LivelinessLostStatus::total_count_change
```

The incremental changes in total_count since the last time the listener was called or the status was read.

9.96 DDS_LivelinessQosPolicy Struct Reference

Specifies and configures the mechanism that allows **DDSDataReader** (p. 1556) entities to detect when **DDSDataWriter** (p. 1589) entities become disconnected or "dead.".

Public Attributes

- **DDS_LivelinessQosPolicyKind kind**

  *<<**basic**>> (p. 201) <<**spec**>> (p. 209) The kind of liveliness desired.*
- struct **DDS_Duration_t lease_duration**

  *<<**basic**>> (p. 201) <<**spec**>> (p. 209) The duration within which a **DDSDataWriter** (p. 1589) must be asserted, or else it is assumed to be not alive.*
- **DDS_Long assertions_per_lease_duration**

  *<<**basic**>> (p. 201) <<**extension**>> (p. 213) The number of assertions a **DDSDataWriter** (p. 1589) will send during a its **DDS_LivelinessQosPolicy::lease_duration** (p. 1182).*

9.96.1 Detailed Description

Specifies and configures the mechanism that allows **DDSDataReader** (p. 1556) entities to detect when **DDSDataWriter** (p. 1589) entities become disconnected or "dead.".

The liveliness status of a **DDSDataWriter** (p. 1589) is used to maintain instance ownership in combination with the setting of the **OWNERSHIP** (p. 612) policy. The application is also informed via **DDSListener** (p. 1794) when an **DDSDataWriter** (p. 1589) is no longer alive.

A **DDSDataWriter** (p. 1589) commits to signalling its liveliness at intervals not to exceed the **DDS_LivelinessQosPolicy::lease_duration** (p. 1182) configured on the **DDSDataWriter** (p. 1589). The rate at which the **DDSDataWriter** (p. 1589) will signal its liveliness is defined by **DDS_LivelinessQosPolicy::assertions_per_lease_duration** (p. 1183).

The **DDSDataReader** (p. 1556) lease_duration specifies the maximum period at which matching DataWriters must have their liveliness asserted.

In addition, in the subscribing application Connext DDS uses an internal thread that wakes up at the period set by the DataReader's lease_duration to see if a DataWriter lease_duration has been violated.

Important: The DataReader lease_duration only configures how often a DataReader will check the liveliness of the matching DataWriters. The DataWriter lease_duration configures the period at which a DataWriter must assert its liveliness and is the value that the DataReader uses to determine if a matching DataWriter is alive or not.

Listeners are used to notify a **DDSDataReader** (p. 1556) of loss of liveliness and **DDSDataWriter** (p. 1589) of violations to the liveliness contract. The on_liveliness_lost() callback is only called *once*, after the first time the lease_duration is exceeded (when the **DDSDataWriter** (p. 1589) first loses liveliness).

This QoS policy can be used during system integration to ensure that applications have been coded to meet design specifications. It can also be used during runtime to detect when systems are performing outside of design specifications. Receiving applications can take appropriate actions in response to disconnected DataWriters.

Entity:

DDSTopic (p. 1885), **DDSDataReader** (p. 1556), **DDSDataWriter** (p. 1589)

Status:

DDS_LIVELINESS_LOST_STATUS (p. 542), **DDS_LivelinessLostStatus** (p. 1179);
DDS_LIVELINESS_CHANGED_STATUS (p. 542), **DDS_LivelinessChangedStatus** (p. 1176);
DDS_REQUESTED_INCOMPATIBLE_QOS_STATUS (p. 540), **DDS_OFFERED_INCOMPATIBLE_QOS_↩ STATUS** (p. 540)

Properties:

RxO (p. 556) = YES
Changeable (p. 556) = **UNTIL ENABLE** (p. 556)

9.96.2 Usage

This policy controls the mechanism and parameters used by RTI Connext to ensure that particular DataWriters on the network are still alive. The liveliness can also affect the ownership of a particular instance, as determined by the **OWNERSHIP** (p. 612) policy.

This policy has several settings to support both data types that are updated periodically as well as those that are changed sporadically. It also allows customisation for different application requirements in terms of the kinds of failures that will be detected by the liveliness mechanism.

The **DDS_AUTOMATIC_LIVELINESS_QOS** (p. 608) liveliness setting is most appropriate for applications that only need to detect failures at the process-level, but not application-logic failures within a process. RTI Connext takes responsibility for renewing the leases at the required rates and thus, as long as the local process where a **DDSDomain↩ Participant** (p. 1619) is running and the link connecting it to remote participants remains connected, the entities within the **DDSDomainParticipant** (p. 1619) will be considered alive. This requires the lowest overhead.

The manual settings (**DDS_MANUAL_BY_PARTICIPANT_LIVELINESS_QOS** (p. 608), **DDS_MANUAL_BY_TOPIC↩ _LIVELINESS_QOS** (p. 608)) require the application on the publishing side to periodically assert the liveliness before the lease expires to indicate the corresponding **DDSEntity** (p. 1731) is still alive. The action can be explicit by calling the **DDSDataWriter::assert_liveliness** (p. 1597) operation or implicit by writing some data.

The two possible manual settings control the granularity at which the application must assert liveliness.

- The setting **DDS_MANUAL_BY_PARTICIPANT_LIVELINESS_QOS** (p. 608) requires only that one **DDSEntity** (p. 1731) within a participant is asserted to be alive to deduce all other **DDSEntity** (p. 1731) objects within the same **DDSDomainParticipant** (p. 1619) are also alive.

- The setting **DDS_MANUAL_BY_TOPIC_LIVELINESS_QOS** (p. 608) requires that at least one instance within the **DDSDataWriter** (p. 1589) is asserted.

Changes in **LIVELINESS** (p. 607) must be detected by the Service with a time-granularity greater or equal to the **DDS_LivelinessQosPolicy::lease_duration** (p. 1182). This ensures that the value of the **DDS_LivelinessChangedStatus** (p. 1176) is updated at least once during each lease_duration and the related Listeners and **DDSWaitSet** (p. 1897) s are notified within a lease_duration from the time the **LIVELINESS** (p. 607) changed.

9.96.3 Compatibility

The value offered is considered compatible with the value requested if and only if the following conditions are met:

- the inequality *offered kind >= requested kind* evaluates to 'TRUE'. For the purposes of this inequality, the values of **DDS_LivelinessQosPolicyKind** (p. 607) kind are considered ordered such that: **DDS_AUTOMATIC_LIVELINESS_QOS** (p. 608) < **DDS_MANUAL_BY_PARTICIPANT_LIVELINESS_QOS** (p. 608) < **DDS_MANUAL_BY_TOPIC_LIVELINESS_QOS** (p. 608).

- the inequality *offered* `lease_duration` *<= requested* `lease_duration` evaluates to **DDS_BOOLEAN_TRUE** (p. 513).

See also

Relationship between registration, liveliness and ownership (p. 1225)

9.96.4 Member Data Documentation

9.96.4.1 kind

DDS_LivelinessQosPolicyKind DDS_LivelinessQosPolicy::kind

<<***basic***>> *(p. 201)* <<***spec***>> *(p. 209)* The kind of liveliness desired.

[default] DDS_AUTOMATIC_LIVELINESS_QOS (p. 608)

Categories

Immutable (p. 333), **Discovery Metadata** (p. 377), **Matching** (p. 382), **Liveliness** (p. 292)

9.96.4.2 lease_duration

```
struct DDS_Duration_t DDS_LivelinessQosPolicy::lease_duration
```

<<***basic***>> *(p. 201)* <<***spec***>> *(p. 209)* The duration within which a **DDSDataWriter** (p. 1589) must be asserted, or else it is assumed to be not alive.

For a DataWriter, the lease_duration specifies a timeout by which liveliness must be asserted for the DataWriter, or the DataWriter will be considered inactive or not alive.

For a DataReader, the lease_duration specifies the maximum period at which the DataReader will check to see if the matching DataWriters are still alive according to the DataWriters lease_duration value.

Important: The DataWriter lease_duration is the value that the DataReader uses to determine if a matching DataWriter is alive or not.

Writer-side filtering is disabled if the writer's lease_duration is set to a finite value.

[default] DDS_DURATION_INFINITE (p. 522)

[range] [0,1 year] or **DDS_DURATION_INFINITE** (p. 522)

Categories

Immutable (p. 333), **Discovery Metadata** (p. 377), **Matching** (p. 382), **Liveliness** (p. 292)

9.96.4.3 assertions_per_lease_duration

```
DDS_Long DDS_LivelinessQosPolicy::assertions_per_lease_duration
```

<<***basic***>> *(p. 201)* <<***extension***>> *(p. 213)* The number of assertions a **DDSDataWriter** (p. 1589) will send during a its **DDS_LivelinessQosPolicy::lease_duration** (p. 1182).

This field only applies to a **DDSDataWriter** (p. 1589) and is not considered during QoS compatibility checks.

The default value is 3. A higher value will make the liveliness mechanism more robust against packet losses, but it will also increase the network traffic.

[default] 3

[range] [2, 100 million]

Categories

Immutable (p. 333), **Liveliness** (p. 292)

9.97 DDS_Locator_t Struct Reference

*<<**extension**>> (p. 213)* Type used to represent the addressing information needed to send a message to an RTPS Endpoint using one of the supported transports.

Public Attributes

- **DDS_Long kind**

  The kind of locator.
- **DDS_UnsignedLong port**

  the port number
- **DDS_Octet address** [**DDS_LOCATOR_ADDRESS_LENGTH_MAX**]

  *A **DDS_LOCATOR_ADDRESS_LENGTH_MAX** (p. 501) octet field to hold the IP address.*

9.97.1 Detailed Description

*<<**extension**>> (p. 213)* Type used to represent the addressing information needed to send a message to an RTPS Endpoint using one of the supported transports.

9.97.2 Member Data Documentation

9.97.2.1 kind

```
DDS_Long DDS_Locator_t::kind
```

The kind of locator.

If the Locator_t kind is **DDS_LOCATOR_KIND_UDPv4** (p. 507), the address contains an IPv4 address. In this case, the leading 12 octets of the **DDS_Locator_t::address** (p. 1184) must be zero. The last 4 octets of **DDS_Locator_t::address** (p. 1184) are used to store the IPv4 address.

If the Locator_t kind is **DDS_LOCATOR_KIND_UDPv6** (p. 507), the address contains an IPv6 address. IPv6 addresses typically use a shorthand hexadecimal notation that maps one-to-one to the 16 octets in the **DDS_Locator_t::address** (p. 1184) field.

9.97.2.2 port

```
DDS_UnsignedLong DDS_Locator_t::port
```

the port number

9.97.2.3 address

```
DDS_Octet DDS_Locator_t::address[ DDS_LOCATOR_ADDRESS_LENGTH_MAX]
```

A **DDS_LOCATOR_ADDRESS_LENGTH_MAX** (p. 501) octet field to hold the IP address.

9.98 DDS_LocatorFilter_t Struct Reference

<<***extension***>> *(p. 213)* Specifies the configuration of an individual channel within a MultiChannel DataWriter.

Public Attributes

- struct **DDS_LocatorSeq locators**

  <<***extension***>> *(p. 213) Sequence containing from one to 16* ***DDS_Locator_t*** *(p. 1184), used to specify the multicast address locators of an individual channel within a MultiChannel DataWriter.*
- char * **filter_expression**

  <<***extension***>> *(p. 213) A logical expression used to determine the data that will be published in the channel.*

9.98.1 Detailed Description

<<***extension***>> *(p. 213)* Specifies the configuration of an individual channel within a MultiChannel DataWriter.

QoS:

DDS_LocatorFilterQosPolicy (p. 1186)

9.98.2 Member Data Documentation

9.98.2.1 locators

```
struct DDS_LocatorSeq DDS_LocatorFilter_t::locators
```

<<***extension***>> *(p. 213)* Sequence containing from one to 16 **DDS_Locator_t** (p. 1184), used to specify the multicast address locators of an individual channel within a MultiChannel DataWriter.

[default] Empty sequence.

9.98.2.2 filter_expression

```
char* DDS_LocatorFilter_t::filter_expression
```

<<***extension***>> *(p. 213)* A logical expression used to determine the data that will be published in the channel.

If the expression evaluates to TRUE, a sample will be published on the channel.

An empty string always evaluates the expression to TRUE.

A NULL value is not allowed.

The syntax of the expression will depend on the value of **DDS_LocatorFilterQosPolicy::filter_name** (p. 1187)

Important: This value must be an allocated string with **DDS_String_alloc** (p. 759) or **DDS_String_dup** (p. 761). It should not be assigned to a string constant.

See also

Queries and Filters Syntax (p. 184)

[default] NULL (invalid value)

9.99 DDS_LocatorFilterQosPolicy Struct Reference

<<***extension***>> *(p. 213)* The QoS policy used to report the configuration of a MultiChannel DataWriter as part of **DDS_PublicationBuiltinTopicData** (p. 1264).

Public Attributes

- struct **DDS_LocatorFilterSeq locator_filters**

  <<***extension***>> *(p. 213) A sequence of **DDS_LocatorFilter_t** (p. 1185). Each **DDS_LocatorFilter_t** (p. 1185) reports the configuration of a single channel of a MultiChannel DataWriter.*
- char * **filter_name**

  <<***extension***>> *(p. 213) Name of the filter class used to describe the filter expressions of a MultiChannel DataWriter.*

9.99.1 Detailed Description

<<***extension***>> *(p. 213)* The QoS policy used to report the configuration of a MultiChannel DataWriter as part of **DDS_PublicationBuiltinTopicData** (p. 1264).

Entity:

DDS_PublicationBuiltinTopicData (p. 1264)

Properties:

RxO (p. 556) = N/A
Changeable (p. 556) = **NO** (p. 556)

9.99.2 Member Data Documentation

9.99.2.1 locator_filters

```
struct DDS_LocatorFilterSeq DDS_LocatorFilterQosPolicy::locator_filters
```

<<***extension***>> *(p. 213)* A sequence of **DDS_LocatorFilter_t** (p. 1185). Each **DDS_LocatorFilter_t** (p. 1185) reports the configuration of a single channel of a MultiChannel DataWriter.

A sequence length of zero indicates the **DDS_MultiChannelQosPolicy** (p. 1214) is not in use.

[default] Empty sequence.

9.99.2.2 filter_name

```
char* DDS_LocatorFilterQosPolicy::filter_name
```

<<***extension***>> *(p. 213)* Name of the filter class used to describe the filter expressions of a MultiChannel DataWriter.

The following builtin filters are supported: **DDS_SQLFILTER_NAME** (p. 61) and **DDS_STRINGMATCHFILTER_NAME** (p. 61).

Warning

This value must be assigned to either one of the predefined values (**DDS_SQLFILTER_NAME** (p. 61) or **DDS_STRINGMATCHFILTER_NAME** (p. 61)), or to an allocated string with **DDS_String_alloc** (p. 759) or **DDS_String_dup** (p. 761). It should not be assigned to a string constant.

[default] DDS_STRINGMATCHFILTER_NAME (p. 61)

9.100 DDS_LocatorFilterSeq Struct Reference

Declares IDL `sequence<` **DDS_LocatorFilter_t** (p. 1185) `>`.

9.100.1 Detailed Description

Declares IDL `sequence<` **DDS_LocatorFilter_t** (p. 1185) `>`.

A sequence of **DDS_LocatorFilter_t** (p. 1185) used to report the channels' properties. If the length of the sequence is zero, the **DDS_MultiChannelQosPolicy** (p. 1214) is not in use.

Instantiates:

<<***generic***>> *(p. 433)* **FooSeq** (p. 1965)

See also

DDS_LocatorFilter_t (p. 1185)

9.101 DDS_LocatorSeq Struct Reference

Declares IDL `sequence` < **DDS_Locator_t** (p. 1184) >

9.101.1 Detailed Description

Declares IDL `sequence` < **DDS_Locator_t** (p. 1184) >

See also

DDS_Locator_t (p. 1184)

9.102 DDS_LoggingQosPolicy Struct Reference

<<***extension***>> *(p. 213)* Configures the RTI Connext logging facility.

Public Attributes

- **NDDS_Config_LogVerbosity verbosity**
  <<***basic***>> *(p. 201)* <<***extension***>> *(p. 213) The verbosities at which RTI Connext diagnostic information is logged.*
- **NDDS_Config_LogCategory category**
  <<***basic***>> *(p. 201)* <<***extension***>> *(p. 213) Categories of logged messages.*
- **NDDS_Config_LogPrintFormat print_format**
  <<***extension***>> *(p. 213) The format used to output RTI Connext diagnostic information.*
- char * **output_file**
  <<***basic***>> *(p. 201)* <<***extension***>> *(p. 213) Specifies the file to which log messages will be redirected to.*
- char * **output_file_suffix**
  <<***basic***>> *(p. 201)* <<***extension***>> *(p. 213) Sets the file suffix when logging to a set of files*
- **DDS_Long max_bytes_per_file**
  <<***extension***>> *(p. 213) Specifies the maximum number of bytes a single file can contain*
- **DDS_Long max_files**
  <<***extension***>> *(p. 213) Specifies the maximum number of files to create before overwritting the previous ones.*

9.102.1 Detailed Description

<<***extension***>> *(p. 213)* Configures the RTI Connext logging facility.

All the properties associated with RTI Connext logging can be configured using this QoS policy. This allows you to configure logging using XML QoS Profiles. See the "Troubleshooting" chapter in the `User's Manual` for details.

Entity:

DDSDomainParticipantFactory (p. 1694)

Properties:

RxO (p. 556) = NO
Changeable (p. 556) = **Changeable** (p. 556)

9.102.2 Member Data Documentation

9.102.2.1 verbosity

NDDS_Config_LogVerbosity DDS_LoggingQosPolicy::verbosity

<<***basic***>> *(*p. *201)* <<***extension***>> (p. *213)* The verbosities at which RTI Connext diagnostic information is logged.

[default] NDDS_CONFIG_LOG_VERBOSITY_ERROR (p. 724)

Categories

Mutable (p. 369), **Logging** (p. 294), **Debuggability** (p. 266)

9.102.2.2 category

NDDS_Config_LogCategory DDS_LoggingQosPolicy::category

<<***basic***>> (p. *201)* <<***extension***>> *(*p. *213)* Categories of logged messages.

[default] Logging will be enabled for all the categories.

Categories

Mutable (p. 369), **Logging** (p. 294), **Debuggability** (p. 266)

9.102.2.3 print_format

NDDS_Config_LogPrintFormat DDS_LoggingQosPolicy::print_format

<<***extension***>> *(*p. *213)* The format used to output RTI Connext diagnostic information.

[default] NDDS_CONFIG_LOG_PRINT_FORMAT_DEFAULT (p. 726).

Categories

Mutable (p. 369), **Logging** (p. 294), **Debuggability** (p. 266)

9.102.2.4 output_file

```
char* DDS_LoggingQosPolicy::output_file
```

<<***basic***>> *(p. 201)* <<***extension***>> *(p. 213)* Specifies the file to which log messages will be redirected to.

If the value of ouput_file is set to NULL, log messages will sent to standard output.

If **DDS_LoggingQosPolicy::max_bytes_per_file** (p. 1190) is not **DDS_LENGTH_UNLIMITED** (p. 636), this is used as the file name preffix for a set of numbered files.

Important: This value must be an allocated string with **DDS_String_alloc** (p. 759) or **DDS_String_dup** (p. 761). It should not be assigned to a string constant.

[default] NULL

See also

NDDSConfigLogger::set_output_file_set (p. 2096)

Categories

Mutable (p. 369), **Logging** (p. 294), **Debuggability** (p. 266)

9.102.2.5 output_file_suffix

```
char* DDS_LoggingQosPolicy::output_file_suffix
```

<<***basic***>> *(p. 201)* <<***extension***>> *(p. 213)* Sets the file suffix when logging to a set of files

Note

This field only applies when idref_LoggingQosPolicy_max_bytes_per_file is different than **DDS_LENGTH_↩ UNLIMITED** (p. 636).

It specifies the suffix to use for the set of files used to redirect the logging output. The preffix is **DDS_LoggingQos↩ Policy::output_file** (p. 1189).

Important: This value must be an allocated string with **DDS_String_alloc** (p. 759) or **DDS_String_dup** (p. 761). It should not be assigned to a string constant.

[default] NULL

[default] No suffix

See also

NDDSConfigLogger::set_output_file_set (p. 2096)

Categories

Mutable (p. 369), **Logging** (p. 294), **Debuggability** (p. 266)

9.102.2.6 max_bytes_per_file

```
DDS_Long DDS_LoggingQosPolicy::max_bytes_per_file
```

<<***extension***>> *(p. 213)* Specifies the maximum number of bytes a single file can contain

When this field is different than **DDS_LENGTH_UNLIMITED** (p. 636), it enables logging to separate files as they reach this size.

[default] DDS_LENGTH_UNLIMITED (p. 636) (a single file is used)

See also

NDDSConfigLogger::set_output_file_set (p. 2096)

Categories

Mutable (p. 369), **Logging** (p. 294), **Debuggability** (p. 266), **Middleware Memory Management** (p. 297)

9.102.2.7 max_files

```
DDS_Long DDS_LoggingQosPolicy::max_files
```

<<***extension***>> *(p. 213)* Specifies the maximum number of files to create before overwritting the previous ones.

Note

This field only applies when idref_LoggingQosPolicy_max_bytes_per_file is differenet than **DDS_LENGTH_↩ UNLIMITED** (p. 636).

When this field is different than **DDS_LENGTH_UNLIMITED** (p. 636), and the number of files reaches this number, future logging messages overwrite the previously created files.

[default] DDS_LENGTH_UNLIMITED (p. 636) (files aren't overwritten)

See also

NDDSConfigLogger::set_output_file_set (p. 2096)

Categories

Mutable (p. 369), **Logging** (p. 294), **Debuggability** (p. 266), **Middleware Memory Management** (p. 297)

9.103 DDS_LongDoubleSeq Struct Reference

Instantiates **FooSeq** (p. 1965) < **DDS_LongDouble** (p. 516) >

9.103.1 Detailed Description

Instantiates **FooSeq** (p. 1965) < **DDS_LongDouble** (p. 516) >

Instantiates:

<<*generic*>> *(*p. *433)* **FooSeq** (p. 1965)

See also

DDS_LongDouble (p. 516)

FooSeq (p. 1965)

9.104 DDS_LongLongSeq Struct Reference

Instantiates **FooSeq** (p. 1965) < **DDS_LongLong** (p. 515) >

9.104.1 Detailed Description

Instantiates **FooSeq** (p. 1965) < **DDS_LongLong** (p. 515) >

Instantiates:

<<*generic*>> *(*p. *433)* **FooSeq** (p. 1965)

See also

DDS_LongLong (p. 515)

FooSeq (p. 1965)

9.105 DDS_LongSeq Struct Reference

Instantiates **FooSeq** (p. 1965) < **DDS_Long** (p. 515) >

9.105.1 Detailed Description

Instantiates **FooSeq** (p. 1965) < **DDS_Long** (p. 515) >

Instantiates:

<<***generic***>> *(p. 433)* **FooSeq** (p. 1965)

See also

DDS_Long (p. 515)

FooSeq (p. 1965)

9.106 DDS_MonitoringDedicatedParticipantSettings Struct Reference

<<***extension***>> *(p. 213)* Configures the use of a dedicated **DDSDomainParticipant** (p. 1619) to distribute the RTI Connext application telemetry data.

Public Attributes

- **DDS_Boolean enable**

  <<***basic***>> *(p. 201)* <<***extension***>> *(p. 213) Enables the use of a dedicated **DDSDomainParticipant** (p. 1619) to distribute the RTI Connext application telemetry data.*
- int **domain_id**

  <<***basic***>> *(p. 201)* <<***extension***>> *(p. 213) The domain ID used in the creation of RTI Monitoring Library 2.0 **DDSDomainParticipant** (p. 1619).*
- char * **participant_qos_profile_name**

  <<***extension***>> *(p. 213) The fully qualified name of the profile used to configure the **DDSDomainParticipant** (p. 1619) that will be used to distribute telemetry data.*
- struct **DDS_StringSeq collector_initial_peers**

  <<***basic***>> *(p. 201)* <<***extension***>> *(p. 213) Determines the initial list of peers that the discovery process will contact to send announcements about the presence of the **DDS_MonitoringDistributionSettings::dedicated_participant** (p. 1196).*

9.106.1 Detailed Description

<<***extension***>> *(p. 213)* Configures the use of a dedicated **DDSDomainParticipant** (p. 1619) to distribute the RTI Connext application telemetry data.

9.106.2 Member Data Documentation

9.106.2.1 enable

```
DDS_Boolean DDS_MonitoringDedicatedParticipantSettings::enable
```

<<***basic***>> *(*p. *201)* <<***extension***>> *(*p. *213)* Enables the use of a dedicated **DDSDomainParticipant** (p. 1619) to distribute the RTI Connext application telemetry data.

Setting this value to **DDS_BOOLEAN_FALSE** (p. 513) is not currently supported.

[default] DDS_BOOLEAN_TRUE (p. 513)

Categories

Mutable (p. 369), **Debuggability** (p. 266), **QoS Profile Management** (p. 312)

9.106.2.2 domain_id

```
int DDS_MonitoringDedicatedParticipantSettings::domain_id
```

<<***basic***>> *(*p. *201)* <<***extension***>> *(*p. *213)* The domain ID used in the creation of RTI Monitoring Library 2.0 **DDSDomainParticipant** (p. 1619).

By default, the RTI Monitoring Library 2.0 DomainParticipant is created in domain 101 with the domain tag 'RTI_o11y'. The domain tag provides further isolation in case the application also uses domain 101.

The domain ID can be changed using this field. Changing the domain tag is not recommended. However, if you need to change it, you must do so by configuring the domain tag in the **DDS_DomainParticipantQos** (p. 971) of the profile set in **DDS_MonitoringDedicatedParticipantSettings::participant_qos_profile_name** (p. 1194).

[default] 101

Categories

Mutable (p. 369), **Debuggability** (p. 266), **Discovery** (p. 271), **Discovery Isolation** (p. 277)

9.106.2.3 participant_qos_profile_name

```
char* DDS_MonitoringDedicatedParticipantSettings::participant_qos_profile_name
```

<<***extension***>> *(*p. *213)* The fully qualified name of the profile used to configure the **DDSDomainParticipant** (p. 1619) that will be used to distribute telemetry data.

If NULL (the default value) then RTI Monitoring Library 2.0 uses **DDS_PROFILE_GENERIC_MONITORING2** (p. 695).

[default] NULL

Categories

Mutable (p. 369), **Debuggability** (p. 266), **QoS Profile Management** (p. 312)

9.106.2.4 collector_initial_peers

```
struct DDS_StringSeq DDS_MonitoringDedicatedParticipantSettings::collector_initial_peers
```

<<***basic***>> *(p. 201)* <<***extension***>> *(p. 213)* Determines the initial list of peers that the discovery process will contact to send announcements about the presence of the **DDS_MonitoringDistributionSettings::dedicated_participant** (p. 1196).

These initial peers should correspond with the RTI Observability Collector Service with which RTI Monitoring Library 2.0 has to communicate. The collector_initial_peers works the same as initial_peers for other DomainParticipants, except that it allows you to easily specify the initial peer(s) for the RTI Monitoring Library 2.0 **DDSDomainParticipant** (p. 1619), which may have different initial peer(s) than those used by your application.

If no collector_initial_peers are specified, or if it is explicitly set to an empty list, the **DDS_DiscoveryQosPolicy::initial_peers** (p. 962) list of **DDS_MonitoringDedicatedParticipantSettings::participant_qos_profile_name** (p. 1194) will be used as the initial peers of **DDS_MonitoringDistributionSettings::dedicated_participant** (p. 1196).

[default] An empty sequence.

See also

DDS_DiscoveryQosPolicy::initial_peers (p. 962) for further information about initial peers.

Categories

Mutable (p. 369), **Debuggability** (p. 266), **Discovery** (p. 271), **Discovery Isolation** (p. 277)

9.107 DDS_MonitoringDistributionSettings Struct Reference

<<***extension***>> *(p. 213)* Configures the distribution of telemetry data.

Public Attributes

- struct **DDS_MonitoringDedicatedParticipantSettings dedicated_participant**

  <<***extension***>> *(p. 213) Configures the use of a dedicated **DDSDomainParticipant** (p. 1619) to distribute the RTI Connext application telemetry data.*
- char * **publisher_qos_profile_name**

  <<***extension***>> *(p. 213) The fully qualified name of the profile used to configure the Publishers that distribute telemetry data.*
- struct **DDS_MonitoringEventDistributionSettings event_settings**

  <<***extension***>> *(p. 213) Configures the distribution of event metrics.*
- struct **DDS_MonitoringPeriodicDistributionSettings periodic_settings**

  <<***extension***>> *(p. 213) Configures the distribution of periodic metrics.*
- struct **DDS_MonitoringLoggingDistributionSettings logging_settings**

  <<***extension***>> *(p. 213) Configures the distribution of logging messages.*

9.107.1 Detailed Description

<<***extension***>> *(p. 213)* Configures the distribution of telemetry data.

9.107.2 Member Data Documentation

9.107.2.1 dedicated_participant

`struct` **`DDS_MonitoringDedicatedParticipantSettings`** `DDS_MonitoringDistributionSettings::dedicated_participant`

<<***extension***>> *(p. 213)* Configures the use of a dedicated **DDSDomainParticipant** (p. 1619) to distribute the RTI Connext application telemetry data.

Categories

Mutable (p. 369), **Debuggability** (p. 266), **QoS Profile Management** (p. 312)

9.107.2.2 publisher_qos_profile_name

`char* DDS_MonitoringDistributionSettings::publisher_qos_profile_name`

<<***extension***>> *(p. 213)* The fully qualified name of the profile used to configure the Publishers that distribute telemetry data.

There is one Publisher for each telemetry data **DDSTopic** (p. 1885): **RTI_MONITORING_PERIODIC_TOPIC_NAME** (p. 720), **RTI_MONITORING_EVENT_TOPIC_NAME** (p. 720), and **RTI_MONITORING_LOGGING_TOPIC_NAME** (p. 721).

If NULL (the default value) then RTI Monitoring Library 2.0 uses **DDS_PROFILE_GENERIC_MONITORING2** (p. 695).

[default] NULL

Categories

Mutable (p. 369), **Debuggability** (p. 266), **QoS Profile Management** (p. 312)

9.107.2.3 event_settings

`struct` **`DDS_MonitoringEventDistributionSettings`** `DDS_MonitoringDistributionSettings::event_settings`

<<***extension***>> *(p. 213)* Configures the distribution of event metrics.

Categories

Mutable (p. 369), **Debuggability** (p. 266)

9.107.2.4 periodic_settings

`struct` **`DDS_MonitoringPeriodicDistributionSettings`** `DDS_MonitoringDistributionSettings::periodic_↩settings`

<<***extension***>> *(p. 213)* Configures the distribution of periodic metrics.

Categories

Mutable (p. 369), **Debuggability** (p. 266)

9.107.2.5 logging_settings

`struct` **`DDS_MonitoringLoggingDistributionSettings`** `DDS_MonitoringDistributionSettings::logging_↩settings`

<<***extension***>> *(p. 213)* Configures the distribution of logging messages.

Categories

Mutable (p. 369), **Debuggability** (p. 266), **Logging** (p. 294)

9.108 DDS_MonitoringEventDistributionSettings Struct Reference

<<***extension***>> *(p. 213)* Configures the distribution of event metrics.

Public Attributes

- **DDS_UnsignedLong concurrency_level**

  <<***extension***>> *(p. 213) Defines how concurrent the push of event metrics to RTI Monitoring Library 2.0 is.*
- char * **datawriter_qos_profile_name**

  <<***extension***>> *(p. 213) The fully qualified name of the profile used to configure the* ***DDSDataWriter*** *(p. 1589) that distributes event metrics.*
- struct **DDS_ThreadSettings_t thread**

  <<***extension***>> *(p. 213) The settings of the event metric thread.*
- struct **DDS_Duration_t publication_period**

  <<***basic***>> *(p. 201)* <<***extension***>> *(p. 213) Period at which the event metric thread publishes the event metrics that have changed since the last time they were published.*

9.108.1 Detailed Description

<<***extension***>> *(p. 213)* Configures the distribution of event metrics.

Event metrics are provided to RTI Monitoring Library 2.0 when they change.

For example, if the liveliness of a **DDSDataWriter** (p. 1589) is lost, a matching **DDSDataReader** (p. 1556) will push the new value of **DDS_LivelinessChangedStatus** (p. 1176) to RTI Monitoring Library 2.0 so that the liveliness status change can be distributed.

There are three kinds of event metrics:

- **Configuration metrics:** Provided to RTI Monitoring Library 2.0 by pushing changes to QoS policies.
- **Status metrics:** Provided to RTI Monitoring Library 2.0 by pushing changes to the event statuses such as **DDS_LivelinessChangedStatus** (p. 1176).
- **Resource metrics:** Provided to RTI Monitoring Library 2.0 when a resource (such as **DDSDataWriter** (p. 1589)) is created or deleted.

The event metrics that will be distributed for an observable resource can be configured with **DDS_MonitoringTelemetryData::metrics** (p. 1213).

9.108.2 Member Data Documentation

9.108.2.1 concurrency_level

```
DDS_UnsignedLong DDS_MonitoringEventDistributionSettings::concurrency_level
```

<<***extension***>> *(p. 213)* Defines how concurrent the push of event metrics to RTI Monitoring Library 2.0 is.

With a concurrency_level of one, all the event metrics pushed to RTI Monitoring Library 2.0 will be stored in a single queue protected by a single mutex.

With a concurrency_level of 'n', RTI Monitoring Library 2.0 will create 'n' queues for event metrics, each queue protected by its own mutex. Each resource (e.g, a **DDSDataReader** (p. 1556)) will be associated with one of the queues when the resource is registered with RTI Monitoring Library 2.0. Therefore, all the event metrics for a single resource always go to the same queue.

The event metrics for two resources associated with different event queues can be pushed in parallel. This is why a higher concurrency_level provides more concurrency.

The event metrics added to the event queues are processed by a single thread configured using **DDS_Monitoring↩ EventDistributionSettings::thread** (p. 1199).

[default] 5

[range] [1, 100]

Categories

Mutable (p. 369), **Concurrency** (p. 258), **Debuggability** (p. 266)

9.108.2.2 datawriter_qos_profile_name

```
char* DDS_MonitoringEventDistributionSettings::datawriter_qos_profile_name
```

<<***extension***>> *(p. 213)* The fully qualified name of the profile used to configure the **DDSDataWriter** (p. 1589) that distributes event metrics.

The **DDSDataWriter** (p. 1589) Topic is **RTI_MONITORING_EVENT_TOPIC_NAME** (p. 720).

If NULL (the default value), then RTI Monitoring Library 2.0 uses **DDS_PROFILE_GENERIC_MONITORING2** (p. 695).

[default] NULL

Categories

Mutable (p. 369), **Debuggability** (p. 266), **QoS Profile Management** (p. 312)

9.108.2.3 thread

`struct` **`DDS_ThreadSettings_t`** `DDS_MonitoringEventDistributionSettings::thread`

<<***extension***>> *(p. 213)* The settings of the event metric thread.

The event metric thread periodically publishes the event metrics pushed into RTI Monitoring Library 2.0 event metric queues after they change their values.

The thread runs at the period configured using **DDS_MonitoringEventDistributionSettings::publication_period** (p. 1200).

[default] DDS_THREAD_SETTINGS_DEFAULT

Categories

Mutable (p. 369), **Debuggability** (p. 266), **Concurrency** (p. 258)

9.108.2.4 publication_period

`struct` **`DDS_Duration_t`** `DDS_MonitoringEventDistributionSettings::publication_period`

<<***basic***>> *(p. 201)* <<***extension***>> *(p. 213)* Period at which the event metric thread publishes the event metrics that have changed since the last time they were published.

With a period of 0 seconds, changes to event metrics will be published immediately after they are pushed into RTI Monitoring Library 2.0.

[default] 1 second

Categories

Mutable (p. 369), **Debuggability** (p. 266), **Data Availability** (p. 261)

9.109 DDS_MonitoringLoggingDistributionSettings Struct Reference

<<***extension***>> *(p. 213)* Configures the distribution of log messages.

Public Attributes

- **DDS_UnsignedLong concurrency_level**

  *<<**extension**>> (p. 213) Defines how concurrent the push of log messages to RTI Monitoring Library 2.0 is.*
- **DDS_UnsignedLong max_historical_logs**

  *<<**basic**>> (p. 201) <<**extension**>> (p. 213) The number of log messages that RTI Monitoring Library 2.0 will keep as history.*
- char * **datawriter_qos_profile_name**

  *<<**extension**>> (p. 213) The fully qualified name of the profile used to configure the **DDSDataWriter** (p. 1589) that distributes log messages.*
- struct **DDS_ThreadSettings_t thread**

  *<<**extension**>> (p. 213) The settings of the logging thread.*
- struct **DDS_Duration_t publication_period**

  *<<**basic**>> (p. 201) <<**extension**>> (p. 213) Period at which the logging thread publishes log messages.*

9.109.1 Detailed Description

*<<**extension**>> (p. 213)* Configures the distribution of log messages.

Log messages are pushed into RTI Monitoring Library 2.0 and published by the logging thread.

The logging thread only publishes a log message with a Syslog level smaller than or equal to the forwarding level of the **NDDS_Config_LogFacility** (p. 727) associated with the log message.

The default value of the forwarding level for all facilities is **NDDS_CONFIG_SYSLOG_VERBOSITY_WARNING** (p. 728). This value can be changed with the **DDS_MonitoringTelemetryData::logs** (p. 1213) QoS Policy or by sending a command to RTI Monitoring Library 2.0.

In this release, commands can only be sent from the RTI Observability Dashboards.

RTI Monitoring Library 2.0 can be configured to keep a history of log messages for later distribution when a log snapshot is requested by a RTI Observability Collector Service.

The log messages maintained in the history are the last 'n' messages published by the logging thread (where 'n' is the value of **DDS_MonitoringLoggingDistributionSettings::max_historical_logs** (p. 1202)).

9.109.2 Member Data Documentation

9.109.2.1 concurrency_level

`DDS_UnsignedLong DDS_MonitoringLoggingDistributionSettings::concurrency_level`

<<***extension***>> *(p. 213)* Defines how concurrent the push of log messages to RTI Monitoring Library 2.0 is.

With a concurrency_level of one, all the log messages pushed to RTI Monitoring Library 2.0 will be stored into a single queue protected by a single mutex.

With a concurrency_level of 'n', RTI Monitoring Library 2.0 will create 'n' queues for log messages, each queue protected by its own mutex.

The log messages generated by a single thread will always be pushed to the same queue. The log messages for two threads associated with different log queues can be pushed in parallel. This is why a higher concurrency_level provides more concurrency.

The log messages added to the log queues are processed and published by a single thread configured using **DDS_↩ MonitoringLoggingDistributionSettings::thread** (p. 1203).

[default] 5

[range] [1, 100]

Categories

Mutable (p. 369), **Debuggability** (p. 266), **Logging** (p. 294), **Concurrency** (p. 258)

9.109.2.2 max_historical_logs

`DDS_UnsignedLong DDS_MonitoringLoggingDistributionSettings::max_historical_logs`

<<***basic***>> *(p. 201)* <<***extension***>> (p. *213)* The number of log messages that RTI Monitoring Library 2.0 will keep as history.

RTI Monitoring Library 2.0 will keep as history the last `max_historical_logs` published messages.

A value of 0 means that RTI Monitoring Library 2.0 should not keep any history.

[default] 128

Categories

Mutable (p. 369), **Debuggability** (p. 266), **Logging** (p. 294), **Middleware Memory Management** (p. 297)

9.109.2.3 datawriter_qos_profile_name

```
char* DDS_MonitoringLoggingDistributionSettings::datawriter_qos_profile_name
```

<<***extension***>> *(p. 213)* The fully qualified name of the profile used to configure the **DDSDataWriter** (p. 1589) that distributes log messages.

The **DDSDataWriter** (p. 1589) Topic is **RTI_MONITORING_LOGGING_TOPIC_NAME** (p. 721).

If NULL (the default value), then RTI Monitoring Library 2.0 uses **DDS_PROFILE_GENERIC_MONITORING2** (p. 695).

[default] NULL

Categories

Mutable (p. 369), **Debuggability** (p. 266), **QoS Profile Management** (p. 312), **Logging** (p. 294)

9.109.2.4 thread

```
struct DDS_ThreadSettings_t DDS_MonitoringLoggingDistributionSettings::thread
```

<<***extension***>> *(p. 213)* The settings of the logging thread.

The logging thread periodically publishes the log messages pushed into RTI Monitoring Library 2.0 log message queues after they are generated.

The thread runs at the period configured using **DDS_MonitoringLoggingDistributionSettings::publication_period** (p. 1203).

[default] DDS_THREAD_SETTINGS_DEFAULT

Categories

Mutable (p. 369), **Debuggability** (p. 266), **Concurrency** (p. 258), **Logging** (p. 294)

9.109.2.5 publication_period

```
struct DDS_Duration_t DDS_MonitoringLoggingDistributionSettings::publication_period
```

<<***basic***>> *(p. 201)* <<***extension***>> *(p. 213)* Period at which the logging thread publishes log messages.

With a period of 0 seconds, log messages will be published immediately after they are pushed into RTI Monitoring Library 2.0.

[default] 1 second

Categories

Mutable (p. 369), **Debuggability** (p. 266), **Data Availability** (p. 261), **Logging** (p. 294)

9.110 DDS_MonitoringLoggingForwardingSettings Struct Reference

<<***extension***>> *(p. 213)* Configures the forwarding levels of log messages for the different **NDDS_Config_Log↩ Facility** (p. 727).

Public Attributes

- **NDDS_Config_SyslogVerbosity middleware_forwarding_level**

  <<***basic***>> *(p. 201)* <<***extension***>> *(p. 213) Log messages with **NDDS_CONFIG_LOG_FACILITY_MIDDLEWARE** (p. 727) and a log level more verbose than this value will not be distributed by RTI Monitoring Library 2.0.*
- **NDDS_Config_SyslogVerbosity security_event_forwarding_level**

  <<***basic***>> *(p. 201)* <<***extension***>> *(p. 213) Log messages with **NDDS_CONFIG_LOG_FACILITY_SECURITY_↩ EVENT** (p. 727) and a log level more verbose than this value will not be distributed by RTI Monitoring Library 2.0.*
- **NDDS_Config_SyslogVerbosity service_forwarding_level**

  <<***basic***>> *(p. 201)* <<***extension***>> *(p. 213) Log messages with **NDDS_CONFIG_LOG_FACILITY_SERVICE** (p. 727) and a log level more verbose than this value will not be distributed by RTI Monitoring Library 2.0.*
- **NDDS_Config_SyslogVerbosity user_forwarding_level**

  <<***basic***>> *(p. 201)* <<***extension***>> *(p. 213) Log messages with **NDDS_CONFIG_LOG_FACILITY_USER** (p. 727) and a log level more verbose than this value will not be distributed by RTI Monitoring Library 2.0.*

9.110.1 Detailed Description

<<***extension***>> *(p. 213)* Configures the forwarding levels of log messages for the different **NDDS_Config_Log↩ Facility** (p. 727).

9.110.2 Member Data Documentation

9.110.2.1 middleware_forwarding_level

`NDDS_Config_SyslogVerbosity` `DDS_MonitoringLoggingForwardingSettings::middleware_forwarding_level`

<<***basic***>> *(p. 201)* <<***extension***>> *(p. 213)* Log messages with **NDDS_CONFIG_LOG_FACILITY_↩ MIDDLEWARE** (p. 727) and a log level more verbose than this value will not be distributed by RTI Monitoring Library 2.0.

[default] NDDS_CONFIG_SYSLOG_VERBOSITY_WARNING (p. 728)

Keep in mind that setting this property beyond WARNING if your application's verbosity is greater than **NDDS_↩ CONFIG_LOG_VERBOSITY_WARNING** (p. 724), may affect performance due to the large amount of messages produced.

Categories

Mutable (p. 369), **Debuggability** (p. 266), **Logging** (p. 294)

9.110.2.2 security_event_forwarding_level

NDDS_Config_SyslogVerbosity DDS_MonitoringLoggingForwardingSettings::security_event_forwarding_level

*<<**basic**>> (p. 201) <<**extension**>> (p. 213)* Log messages with **NDDS_CONFIG_LOG_FACILITY_SECURITY_EVENT** (p. 727) and a log level more verbose than this value will not be distributed by RTI Monitoring Library 2.0.

[default] NDDS_CONFIG_SYSLOG_VERBOSITY_WARNING (p. 728)

Keep in mind that setting this property beyond WARNING if your application's verbosity is greater than **NDDS_CONFIG_LOG_VERBOSITY_WARNING** (p. 724), may affect performance due to the large amount of messages produced.

Categories

Mutable (p. 369), **Debuggability** (p. 266), **Logging** (p. 294), **Security** (p. 325)

9.110.2.3 service_forwarding_level

NDDS_Config_SyslogVerbosity DDS_MonitoringLoggingForwardingSettings::service_forwarding_level

*<<**basic**>> (p. 201) <<**extension**>> (p. 213)* Log messages with **NDDS_CONFIG_LOG_FACILITY_SERVICE** (p. 727) and a log level more verbose than this value will not be distributed by RTI Monitoring Library 2.0.

[default] NDDS_CONFIG_SYSLOG_VERBOSITY_WARNING (p. 728)

Keep in mind that setting this property beyond WARNING if your application's verbosity is greater than **NDDS_CONFIG_LOG_VERBOSITY_WARNING** (p. 724), may affect performance due to the large amount of messages produced.

Categories

Mutable (p. 369), **Debuggability** (p. 266), **Logging** (p. 294)

9.110.2.4 user_forwarding_level

NDDS_Config_SyslogVerbosity DDS_MonitoringLoggingForwardingSettings::user_forwarding_level

*<<**basic**>> (p. 201) <<**extension**>> (p. 213)* Log messages with **NDDS_CONFIG_LOG_FACILITY_USER** (p. 727) and a log level more verbose than this value will not be distributed by RTI Monitoring Library 2.0.

[default] NDDS_CONFIG_SYSLOG_VERBOSITY_WARNING (p. 728)

Keep in mind that setting this property beyond WARNING if your application's verbosity is greater than **NDDS_CONFIG_LOG_VERBOSITY_WARNING** (p. 724), may affect performance due to the large amount of messages produced.

Categories

Mutable (p. 369), **Debuggability** (p. 266), **Logging** (p. 294)

9.111 DDS_MonitoringMetricSelection Struct Reference

<<***extension***>> *(*p. *213)* This data structure is used to configure event and periodic metrics collection and distribution for a specific set of observable resources.

Public Attributes

- char * **resource_selection**

  <<***extension***>> *(*p. *213) An expression pattern that selects a subset of resources by matching the pattern to the resource names.* ***DDS_MonitoringMetricSelection::enabled_metrics_selection*** *(*p. *1207) and* ***DDS_Monitoring↩ MetricSelection::disabled_metrics_selection*** *(*p. *1207) are applied to this subset of resources.*
- struct **DDS_StringSeq enabled_metrics_selection**

  <<***extension***>> *(*p. *213) A sequence of POSIX fnmatch patterns that match the names of the metrics that should be collected and distributed for the observable resources selected by* ***DDS_MonitoringMetricSelection::resource_↩ selection*** *(*p. *1206).*
- struct **DDS_StringSeq disabled_metrics_selection**

  <<***extension***>> *(*p. *213) A sequence of POSIX fnmatch patterns that mach the names of the metrics that should not be collected and distributed for the observable resources selected by* ***DDS_MonitoringMetricSelection::resource_↩ selection*** *(*p. *1206).*

9.111.1 Detailed Description

<<***extension***>> *(*p. *213)* This data structure is used to configure event and periodic metrics collection and distribution for a specific set of observable resources.

9.111.2 Member Data Documentation

9.111.2.1 resource_selection

```
char* DDS_MonitoringMetricSelection::resource_selection
```

<<***extension***>> *(*p. *213)* An expression pattern that selects a subset of resources by matching the pattern to the resource names. **DDS_MonitoringMetricSelection::enabled_metrics_selection** (p. 1207) and **DDS_Monitoring↩ MetricSelection::disabled_metrics_selection** (p. 1207) are applied to this subset of resources.

Examples of resource expression patterns:

- /applications/myApp/domain_participants/myParticipant
- /applications/*/domain_participants/*/subscribers/*
- /applications/myApp/domain_participants/GUID(1234.5678.4321.8765)
- //myEntity

The first expression refers to a DomainParticipant named "myParticipant" that belongs to an application named "myApp". The second expression applies to all the Subscribers in the system (resource name wildcards follow the POSIX fnmatch syntax). The third expression refers to a DomainParticipant with a specific resource GUID in the "myApp" application. The last expression uses the XPath "//" operator. It matches observable resources named "myEntity" no matter where they are located in the resource hieararchy.

See the Telemetry Data / Resources chapter of the RTI Connext Observability Framework documentation for further information on the observable resource names and expression patterns.

Categories

Mutable (p. 369), **Debuggability** (p. 266)

9.111.2.2 enabled_metrics_selection

`struct` **`DDS_StringSeq`** `DDS_MonitoringMetricSelection::enabled_metrics_selection`

<<***extension***>> *(p. 213)* A sequence of POSIX fnmatch patterns that match the names of the metrics that should be collected and distributed for the observable resources selected by **DDS_MonitoringMetricSelection::resource_selection** (p. 1206).

This sequence is evaluated first, followed by **DDS_MonitoringMetricSelection::disabled_metrics_selection** (p. 1207). Therefore, if the same metric is enabled and disabled at the same time, the disablement will be the final result.

The patterns in the sequence are evaluated in order.

Examples of valid patterns:

- dds_data_writer_qos_durability_writer_depth
- dds_data_reader_qos_reliability_*
- dds_application_*

The first pattern refers to a specific DataWriter metric (**DDS_DurabilityQosPolicy::writer_depth** (p. 1008)). The second pattern refers to all the DataReader **DDS_ReliabilityQosPolicy** (p. 1295) metrics. The last pattern selects all the available application metrics.

See the Telemetry Data / Metrics chapter of the RTI Connext Observability Framework documentation for further information on the metric names and metric patterns.

Categories

Mutable (p. 369), **Debuggability** (p. 266)

9.111.2.3 disabled_metrics_selection

```
struct DDS_StringSeq DDS_MonitoringMetricSelection::disabled_metrics_selection
```

<<***extension***>> *(p. 213)* A sequence of POSIX fnmatch patterns that mach the names of the metrics that should not be collected and distributed for the observable resources selected by **DDS_MonitoringMetricSelection::resource_selection** (p. 1206).

This sequence is evaluated after **DDS_MonitoringMetricSelection::enabled_metrics_selection** (p. 1207). Therefore, if the same metric is enabled and disabled at the same time, the disablement will be the final result.

The patterns in the sequence are evaluated in order.

Examples of valid patterns:

- dds_data_writer_qos_durability_writer_depth
- dds_data_reader_qos_reliability_*
- dds_application_*

The first pattern refers to a specific DataWriter metric (**DDS_DurabilityQosPolicy::writer_depth** (p. 1008)). The second pattern refers to all the DataReader **DDS_ReliabilityQosPolicy** (p. 1295) metrics. The last pattern selects all the available application metrics.

See the Telemetry Data / Metrics chapter of the RTI Connext Observability Framework documentation for further information on the metric names and metric patterns.

Categories

Mutable (p. 369), **Debuggability** (p. 266)

9.112 DDS_MonitoringMetricSelectionSeq Struct Reference

Declares IDL `sequence` < **DDS_MonitoringMetricSelection** (p. 1206) >

9.112.1 Detailed Description

Declares IDL `sequence` < **DDS_MonitoringMetricSelection** (p. 1206) >

Instantiates:

<<***generic***>> *(p. 433)* **FooSeq** (p. 1965)

See also

DDS_MonitoringMetricSelection (p. 1206)

9.113 DDS_MonitoringPeriodicDistributionSettings Struct Reference

<<***extension***>> *(p. 213)* Configures the distribution of periodic metrics.

Public Attributes

- char * **datawriter_qos_profile_name**

  <<***extension***>> *(p. 213) The fully qualified name of the profile used to configure the* ***DDSDataWriter*** *(p. 1589) that distributes periodic metrics.*
- struct **DDS_ThreadSettings_t thread**

  <<***extension***>> *(p. 213) The settings of the periodic metric thread.*
- struct **DDS_Duration_t polling_period**

  <<***basic***>> *(p. 201)* <<***extension***>> *(p. 213) Period at which the periodic metric thread polls and publishes the periodic metrics.*

9.113.1 Detailed Description

<<***extension***>> *(p. 213)* Configures the distribution of periodic metrics.

Periodic metrics change often, and they are polled and published periodically by a thread created by RTI Monitoring Library 2.0.

RTI Monitoring Library 2.0 obtains periodic metrics by polling the current value of periodic statuses such as **DDS_DataWriterProtocolStatus** (p. 897).

The periodic metrics that will be distributed for an observable resource can be configured with **DDS_MonitoringTelemetryData::metrics** (p. 1213).

9.113.2 Member Data Documentation

9.113.2.1 datawriter_qos_profile_name

```
char* DDS_MonitoringPeriodicDistributionSettings::datawriter_qos_profile_name
```

<<***extension***>> *(p. 213)* The fully qualified name of the profile used to configure the **DDSDataWriter** (p. 1589) that distributes periodic metrics.

The **DDSDataWriter** (p. 1589) Topic is **RTI_MONITORING_PERIODIC_TOPIC_NAME** (p. 720).

If NULL (the default value), then RTI Monitoring Library 2.0 uses **DDS_PROFILE_GENERIC_MONITORING2** (p. 695).

[default] NULL

Categories

Mutable (p. 369), **Debuggability** (p. 266), **QoS Profile Management** (p. 312)

9.113.2.2 thread

`struct` **`DDS_ThreadSettings_t`** `DDS_MonitoringPeriodicDistributionSettings::thread`

<<***extension***>> *(p. 213)* The settings of the periodic metric thread.

The periodic metric thread periodically polls and publishes periodic event metrics.

The thread runs at the period configured using **DDS_MonitoringPeriodicDistributionSettings::polling_period** (p. 1210).

[default] DDS_THREAD_SETTINGS_DEFAULT

Categories

Mutable (p. 369), **Debuggability** (p. 266), **Concurrency** (p. 258)

9.113.2.3 polling_period

`struct` **`DDS_Duration_t`** `DDS_MonitoringPeriodicDistributionSettings::polling_period`

<<***basic***>> *(p. 201)* <<***extension***>> *(p. 213)* Period at which the periodic metric thread polls and publishes the periodic metrics.

[default] 5 seconds

[range] [1 nanosecond, 4294967 seconds (50 days)]

Categories

Mutable (p. 369), **Debuggability** (p. 266), **Data Availability** (p. 261)

9.114 DDS_MonitoringQosPolicy Struct Reference

<<***extension***>> *(p. 213)* Configures the use of RTI Monitoring Library 2.0 to collect and distribute RTI Connext telemetry data.

Public Attributes

- **DDS_Boolean enable**

  <<***basic***>> *(p. 201)* <<***extension***>> *(p. 213) Enables the collection and distribution of telemetry data for an RTI Connext application using RTI Monitoring Library 2.0.*
- char * **application_name**

  <<***basic***>> *(p. 201)* <<***extension***>> *(p. 213) The name of the resource that represents this RTI Connext application.*
- struct **DDS_MonitoringDistributionSettings distribution_settings**

  <<***extension***>> *(p. 213) Configures the distribution of telemetry data.*
- struct **DDS_MonitoringTelemetryData telemetry_data**

  <<***extension***>> *(p. 213) Configures the telemetry data that will be distributed.*

9.114.1 Detailed Description

<<***extension***>> *(p. 213)* Configures the use of RTI Monitoring Library 2.0 to collect and distribute RTI Connext telemetry data.

9.114.2 Member Data Documentation

9.114.2.1 enable

```
DDS_Boolean DDS_MonitoringQosPolicy::enable
```

<<***basic***>> *(p. 201)* <<***extension***>> *(p. 213)* Enables the collection and distribution of telemetry data for an RTI Connext application using RTI Monitoring Library 2.0.

RTI Monitoring Library 2.0 is **automatically enabled** under the following conditions:

- For RTI Connext applications that link with the RTI Connext shared libraries as long as the rtimonitoring2 shared library is in the library search path. There is no need to link with rtimonitoring2.
- For RTI Connext applications that link with the RTI Connext static libraries, including rtimonitoring2, as long as the application calls the API **RTI_Monitoring_initialize** (p. 720).

RTI Monitoring Library 2.0 is **not automatically enabled** for RTI Connext RTI DDS Ping, RTI DDS Spy, RTI Monitor, RTI Admin Console nor RTI Observability Collector Service.

You can override the default behavior and control whether RTI Monitoring Library 2.0 is enabled or disabled in two ways in order of precedence:

- **Environment variable:** Set `RTI_MONITORING2_ENABLE` to `true` or `false`. This is the simplest way to enable or disable the library without modifying your application or its configuration.
- **QoS policy:** Set this field to **DDS_BOOLEAN_TRUE** (p. 513) or **DDS_BOOLEAN_FALSE** (p. 513) programmatically or via XML.

The environment variable should not be changed at runtime. If you need to **enable** or **disable** RTI Monitoring Library 2.0 at runtime, use this field instead.

Note: Enabling and disabling RTI Monitoring Library 2.0 while DDS Entities are being created or deleted is not a safe operation. The entities created while RTI Monitoring Library 2.0 is being enabled may not be monitored.

Categories

Mutable (p. 369), **Debuggability** (p. 266)

9.114.2.2 application_name

```
char* DDS_MonitoringQosPolicy::application_name
```

<<***basic***>> *(p. 201)* <<***extension***>> *(p. 213)* The name of the resource that represents this RTI Connext application.

When this member is set to a value other than NULL , the resource identifier representing this application will be:

/applications/<application_name>

This is the resource identifier that will be used to send commands to this application from the RTI Observability Dashboards.

The application_name should be unique across the RTI Connext system; however, RTI Monitoring Library 2.0 does not currently enforce uniqueness.

When this member is set to NULL , RTI Monitoring Library 2.0 will automatically assign a resource identifier with this format:

/applications/<host_name:process_id:uuid>

[default] NULL

Categories

Mutable (p. 369), **Debuggability** (p. 266)

9.114.2.3 distribution_settings

```
struct DDS_MonitoringDistributionSettings DDS_MonitoringQosPolicy::distribution_settings
```

<<***extension***>> *(p. 213)* Configures the distribution of telemetry data.

Categories

Mutable (p. 369), **Debuggability** (p. 266)

9.114.2.4 telemetry_data

```
struct DDS_MonitoringTelemetryData DDS_MonitoringQosPolicy::telemetry_data
```

<<***extension***>> *(p. 213)* Configures the telemetry data that will be distributed.

Categories

Mutable (p. 369), **Debuggability** (p. 266)

9.115 DDS_MonitoringTelemetryData Struct Reference

<<***extension***>> *(p. 213)* Configures the telemetry data that will be distributed.

Public Attributes

- struct **DDS_MonitoringMetricSelectionSeq metrics**

  <<***basic***>> *(p. 201)* <<***extension***>> *(p. 213) Sequence of* ***DDS_MonitoringMetricSelection*** *(p. 1206) containing the event and periodic metrics that will be collected and distributed for a given set of observable resources.*
- struct **DDS_MonitoringLoggingForwardingSettings logs**

  <<***basic***>> *(p. 201)* <<***extension***>> *(p. 213)* ***DDS_MonitoringLoggingForwardingSettings*** *(p. 1204) containing the* ***NDDS_Config_SyslogVerbosity*** *(p. 727) levels that will be forwarded for the different* ***NDDS_Config_LogFacility*** *(p. 727).*

9.115.1 Detailed Description

<<***extension***>> *(p. 213)* Configures the telemetry data that will be distributed.

9.115.2 Member Data Documentation

9.115.2.1 metrics

```
struct DDS_MonitoringMetricSelectionSeq DDS_MonitoringTelemetryData::metrics
```

<<***basic***>> *(p. 201)* <<***extension***>> *(p. 213)* Sequence of **DDS_MonitoringMetricSelection** (p. 1206) containing the event and periodic metrics that will be collected and distributed for a given set of observable resources.

The different **DDS_MonitoringMetricSelection** (p. 1206) in the sequence are evaluated in order.

[default] An empty sequence, meaning that no metrics will be collected and distributed for any observable resource.

See also

DDS_MonitoringEventDistributionSettings (p. 1197) and **DDS_MonitoringPeriodicDistributionSettings** (p. 1209) for further information on how RTI Monitoring Library 2.0 distributes **metrics** (p. 1213).

Categories

Mutable (p. 369), **Debuggability** (p. 266)

9.115.2.2 logs

`struct` **DDS_MonitoringLoggingForwardingSettings** `DDS_MonitoringTelemetryData::logs`

<<***basic***>> *(p. 201)* <<***extension***>> *(p. 213)* **DDS_MonitoringLoggingForwardingSettings** (p. 1204) containing the **NDDS_Config_SyslogVerbosity** (p. 727) levels that will be forwarded for the different **NDDS_Config_LogFacility** (p. 727).

See also

DDS_MonitoringLoggingDistributionSettings (p. 1200) for further information on how RTI Monitoring Library 2.0 distributes log messages.

Categories

Mutable (p. 369), **Debuggability** (p. 266), **Logging** (p. 294)

9.116 DDS_MultiChannelQosPolicy Struct Reference

<<***extension***>> *(p. 213)* Configures the ability of a DataWriter to send data on different multicast groups (addresses) based on the value of the data.

Public Attributes

- struct **DDS_ChannelSettingsSeq channels**

  <<***extension***>> *(p. 213) A sequence of* ***DDS_ChannelSettings_t*** *(p. 822) used to configure the channels' properties. If the length of the sequence is zero, the QoS policy will be ignored.*
- char * **filter_name**

  <<***extension***>> *(p. 213) Name of the filter class used to describe the filter expressions of a MultiChannel DataWriter.*

9.116.1 Detailed Description

<<***extension***>> *(p. 213)* Configures the ability of a DataWriter to send data on different multicast groups (addresses) based on the value of the data.

This QoS policy is used to partition the data published by a **DDSDataWriter** (p. 1589) across multiple channels. A *channel* is defined by a filter expression and a sequence of multicast locators.

Entity:

DDSDataWriter (p. 1589)

Properties:

RxO (p. 556) = N/A
Changeable (p. 556) = **NO** (p. 556)

9.116.2 Usage

By using this QoS, a **DDSDataWriter** (p. 1589) can be configured to send data to different multicast groups based on the content of the data. Using syntax similar to those used in Content-Based Filters, you can associate different multicast addresses with filter expressions that operate on the values of the fields within the data. When your application's code calls **FooDataWriter::write** (p. 1951), data is sent to any multicast address for which the data passes the filter.

Multi-channel DataWriters can be used to trade off network bandwidth with the unnecessary processing of unwanted data for situations where there are multiple DataReaders that are interested in different subsets of data that come from the same data stream (Topic). For example, in Financial applications, the data stream may be quotes for different stocks at an exchange. Applications usually only want to receive data (quotes) for only a subset of the stocks being traded. In tracking applications, a data stream may carry information on hundreds or thousands of objects being tracked, but again, applications may only be interested in a subset.

The problem is that the most efficient way to deliver data to multiple applications is to use multicast, so that a data value is only sent once on the network for any number of subscribers to the data. However, using multicast, an application will receive *all* of the data sent and not just the data in which it is interested, thus extra CPU time is wasted to throw away unwanted data. With this QoS, you can analyze the data-usage patterns of your applications and optimize network vs. CPU usage by partitioning the data into multiple multicast streams. While network bandwidth is still being conserved by sending data only once using multicast, most applications will only need to listen to a subset of the multicast addresses and receive a reduced amount of unwanted data.

Your system can gain more of the benefits of using multiple multicast groups if your network uses Layer 2 Ethernet switches. Layer 2 switches can be configured to only route multicast packets to those ports that have added membership to specific multicast groups. Using those switches will ensure that only the multicast packets used by applications on a node are routed to the node; all others are filtered-out by the switch.

9.116.3 Member Data Documentation

9.116.3.1 channels

`struct` **`DDS_ChannelSettingsSeq`** `DDS_MultiChannelQosPolicy::channels`

<<***extension***>> *(p. 213)* A sequence of **DDS_ChannelSettings_t** (p. 822) used to configure the channels' properties. If the length of the sequence is zero, the QoS policy will be ignored.

A sequence length of zero indicates the **DDS_MultiChannelQosPolicy** (p. 1214) is not in use.

The sequence length cannot be greater than **DDS_DomainParticipantResourceLimitsQosPolicy::channel_seq_↩ max_length** (p. 997).

[default] Empty sequence.

Categories

Immutable (p. 333), **Discovery Metadata** (p. 377), **Data Availability** (p. 261)

9.116.3.2 filter_name

```
char* DDS_MultiChannelQosPolicy::filter_name
```

<<***extension***>> *(p. 213)* Name of the filter class used to describe the filter expressions of a MultiChannel DataWriter.

The following builtin filters are supported:

- **DDS_SQLFILTER_NAME** (p. 61)
- **DDS_STRINGMATCHFILTER_NAME** (p. 61)

Warning

The value for this field can be one of the constants above or a string allocated with **DDS_String_dup()** (p. 761) to specify your own filter. You should not assign a string literal. When you assign a new value with **DDS_String_dup()** (p. 761), first check if the current value is one of the constants above. If it is, simply replace it; if it's not, you have to release the current string and assign the new one. Here is an example of such an approach:

```
struct DDS_DataWriterQos writer_qos;
char *filter_name = NULL;
DDS_DataWriterQos_initialize(&writer_qos);
...
DDS_Publisher_get_default_datawriter_qos(publisher, &writer_qos);
...
filter_name = writer_qos.multi_channel.filter_name;
if (filter_name == DDS_SQLFILTER_NAME
        || filter_name == DDS_STRINGMATCH_FILTER_NAME) {
    // Since the value was never on the heap, this won't leak memory
    filter_name = NULL;
} else {
    // This means the value is present on the heap
    DDS_String_free(filter_name);
}
// New value using DDS_String_dup() or one of the predefined constants
filter_name = DDS_String_dup("My custom filter name");
...
DDS_DataWriterQos_finalize(&writer_qos);
```

The strings allocated with **DDS_String_dup()** (p. 761) are released when `writer_qos` is finalized.

More information about string handling can be found under **String Conventions** (p. 758).

[default] DDS_STRINGMATCHFILTER_NAME (p. 61)

Categories

Immutable (p. 333), **Discovery Metadata** (p. 377), **Data Availability** (p. 261), **Content Filtering** (p. 260)

9.117 DDS_Octets Struct Reference

Built-in type consisting of a variable-length array of opaque bytes.

Public Member Functions

- **DDS_Octets** ()
  Constructor.
- **DDS_Octets** (int size)
  Constructor that specifies the size of the allocated octets array.
- ∼**DDS_Octets** ()
  Destructor.

Public Attributes

- int **length**
  Number of octets to serialize.
- unsigned char ∗ **value**
  DDS_Octets *(*p. *1217) array value.*

9.117.1 Detailed Description

Built-in type consisting of a variable-length array of opaque bytes.

9.117.2 Constructor & Destructor Documentation

9.117.2.1 DDS_Octets() [1/2]

```
DDS_Octets::DDS_Octets ( )  [inline]
```

Constructor.

The default constructor initializes the newly created object with NULL value, and zero length.

9.117.2.2 DDS_Octets() [2/2]

```
DDS_Octets::DDS_Octets (
            int size )  [inline]
```

Constructor that specifies the size of the allocated octets array.

After this method is called, length is set to zero.

If a memory allocation failure occurs, and the **DDS_Octets** (p. 1217) structure is allocated but the array inside of it cannot be, the array will be NULL.

Parameters

size	<<***in***>> *(p. 433)* Size of the allocated octets array

References **DDS_OctetBuffer_alloc()**, and **value**.

9.117.2.3 ~DDS_Octets()

```
DDS_Octets::~DDS_Octets ( )  [inline]
```

Destructor.

References **DDS_OctetBuffer_free()**, and **value**.

9.117.3 Member Data Documentation

9.117.3.1 length

```
int DDS_Octets::length
```

Number of octets to serialize.

9.117.3.2 value

```
unsigned char* DDS_Octets::value
```

DDS_Octets (p. 1217) array value.

Referenced by **DDS_Octets()**, and **~DDS_Octets()**.

9.118 DDS_OctetSeq Struct Reference

Instantiates **FooSeq** (p. 1965) < **DDS_Octet** (p. 514) >

9.118.1 Detailed Description

Instantiates **FooSeq** (p. 1965) < **DDS_Octet** (p. 514) >

Instantiates:

<<***generic***>> *(*p. *433)* **FooSeq** (p. 1965)

See also

DDS_Octet (p. 514)

FooSeq (p. 1965)

9.119 DDS_OctetsSeq Struct Reference

Instantiates **FooSeq** (p. 1965) < **DDS_Octets** (p. 1217) > .

9.119.1 Detailed Description

Instantiates **FooSeq** (p. 1965) < **DDS_Octets** (p. 1217) > .

Instantiates:

<<***generic***>> *(*p. *433)* **FooSeq** (p. 1965)

See also

DDS_Octets (p. 1217)

9.120 DDS_OfferedDeadlineMissedStatus Struct Reference

DDS_OFFERED_DEADLINE_MISSED_STATUS (p. 539)

Public Attributes

- **DDS_Long total_count**

  Total cumulative count of the number of times the ***DDSDataWriter*** *(*p. *1589) failed to write within its offered deadline.*
- **DDS_Long total_count_change**

  The incremental changes in total_count since the last time the listener was called or the status was read.
- **DDS_InstanceHandle_t last_instance_handle**

  Handle to the last instance in the ***DDSDataWriter*** *(*p. *1589) for which an offered deadline was missed.*

9.120.1 Detailed Description

DDS_OFFERED_DEADLINE_MISSED_STATUS (p. 539)

Entity:

DDSDataWriter (p. 1589)

Listener:

DDSDataWriterListener (p. 1612)

The deadline that the **DDSDataWriter** (p. 1589) has committed through its **DDS_DeadlineQosPolicy** (p. 928) was not respected for a specific instance.

9.120.2 Member Data Documentation

9.120.2.1 total_count

DDS_Long DDS_OfferedDeadlineMissedStatus::total_count

Total cumulative count of the number of times the **DDSDataWriter** (p. 1589) failed to write within its offered deadline.

Missed deadlines accumulate; that is, each deadline period the `total_count` will be incremented by one.

9.120.2.2 total_count_change

DDS_Long DDS_OfferedDeadlineMissedStatus::total_count_change

The incremental changes in total_count since the last time the listener was called or the status was read.

9.120.2.3 last_instance_handle

DDS_InstanceHandle_t DDS_OfferedDeadlineMissedStatus::last_instance_handle

Handle to the last instance in the **DDSDataWriter** (p. 1589) for which an offered deadline was missed.

9.121 DDS_OfferedIncompatibleQosStatus Struct Reference

DDS_OFFERED_INCOMPATIBLE_QOS_STATUS (p. 540)

Public Attributes

- **DDS_Long total_count**

  *Total cumulative number of times the concerned **DDSDataWriter** (p. 1589) discovered a **DDSDataReader** (p. 1556) for the same **DDSTopic** (p. 1885), common partition with a requested QoS that is incompatible with that offered by the **DDSDataWriter** (p. 1589).*
- **DDS_Long total_count_change**

  The incremental changes in total_count since the last time the listener was called or the status was read.
- **DDS_QosPolicyId_t last_policy_id**

  *The **DDS_QosPolicyId_t** (p. 557) of one of the policies that was found to be incompatible the last time an incompatibility was detected.*
- struct **DDS_QosPolicyCountSeq policies**

  *A list containing for each policy the total number of times that the concerned **DDSDataWriter** (p. 1589) discovered a **DDSDataReader** (p. 1556) for the same **DDSTopic** (p. 1885) and common partition with a requested QoS that is incompatible with that offered by the **DDSDataWriter** (p. 1589).*

9.121.1 Detailed Description

DDS_OFFERED_INCOMPATIBLE_QOS_STATUS (p. 540)

Entity:

DDSDataWriter (p. 1589)

Listener:

DDSDataWriterListener (p. 1612)

The qos policy value was incompatible with what was requested.

9.121.2 Member Data Documentation

9.121.2.1 total_count

```
DDS_Long DDS_OfferedIncompatibleQosStatus::total_count
```

Total cumulative number of times the concerned **DDSDataWriter** (p. 1589) discovered a **DDSDataReader** (p. 1556) for the same **DDSTopic** (p. 1885), common partition with a requested QoS that is incompatible with that offered by the **DDSDataWriter** (p. 1589).

9.121.2.2 total_count_change

DDS_Long DDS_OfferedIncompatibleQosStatus::total_count_change

The incremental changes in total_count since the last time the listener was called or the status was read.

9.121.2.3 last_policy_id

DDS_QosPolicyId_t DDS_OfferedIncompatibleQosStatus::last_policy_id

The **DDS_QosPolicyId_t** (p. 557) of one of the policies that was found to be incompatible the last time an incompatibility was detected.

9.121.2.4 policies

struct **DDS_QosPolicyCountSeq** DDS_OfferedIncompatibleQosStatus::policies

A list containing for each policy the total number of times that the concerned **DDSDataWriter** (p. 1589) discovered a **DDSDataReader** (p. 1556) for the same **DDSTopic** (p. 1885) and common partition with a requested QoS that is incompatible with that offered by the **DDSDataWriter** (p. 1589).

9.122 DDS_OwnershipQosPolicy Struct Reference

*<<**spec**>> (p. 209)* Specifies whether it is allowed for multiple **DDSDataWriter** (p. 1589) (s) to write the same instance of the data and if so, how these modifications should be arbitrated.

Public Attributes

- **DDS_OwnershipQosPolicyKind kind**

  *<<**basic**>> (p. 201) <<**spec**>> (p. 209) The kind of ownership*

9.122.1 Detailed Description

<<***spec***>> *(p. 209)* Specifies whether it is allowed for multiple **DDSDataWriter** (p. 1589) (s) to write the same instance of the data and if so, how these modifications should be arbitrated.

Entity:

DDSTopic (p. 1885), **DDSDataReader** (p. 1556), **DDSDataWriter** (p. 1589)

Status:

DDS_OFFERED_INCOMPATIBLE_QOS_STATUS (p. 540), **DDS_REQUESTED_INCOMPATIBLE_QOS_↩ STATUS** (p. 540)

Properties:

RxO (p. 556) = YES
Changeable (p. 556) = **UNTIL ENABLE** (p. 556)

See also

OWNERSHIP_STRENGTH (p. 613)

9.122.2 Usage

Along with the **OWNERSHIP_STRENGTH** (p. 613), this QoS policy specifies if **DDSDataReader** (p. 1556) entities can receive updates to the same instance (identified by its key) from multiple **DDSDataWriter** (p. 1589) entities at the same time.

There are two kinds of ownership, selected by the setting of the `kind`: SHARED and EXCLUSIVE.

9.122.2.1 SHARED ownership

DDS_SHARED_OWNERSHIP_QOS (p. 613) indicates that RTI Connext does not enforce unique ownership for each instance. In this case, multiple writers can update the same data type instance. The subscriber to the **DDSTopic** (p. 1885) will be able to access modifications from all **DDSDataWriter** (p. 1589) objects, subject to the settings of other QoS that may filter particular samples (e.g. the **TIME_BASED_FILTER** (p. 639) or **HISTORY** (p. 603) policy). In any case, there is no "filtering" of modifications made based on the identity of the **DDSDataWriter** (p. 1589) that causes the modification.

9.122.2.2 EXCLUSIVE ownership

DDS_EXCLUSIVE_OWNERSHIP_QOS (p. 613) indicates that each instance of a data type can only be modified by one **DDSDataWriter** (p. 1589). In other words, at any point in time, a single **DDSDataWriter** (p. 1589) owns each instance and is the only one whose modifications will be visible to the **DDSDataReader** (p. 1556) objects. The owner is determined by selecting the **DDSDataWriter** (p. 1589) with the highest value of the **DDS_OwnershipStrengthQos↩ Policy::value** (p. 1228) that is currently alive, as defined by the **LIVELINESS** (p. 607) policy, and has not violated its **DEADLINE** (p. 583) contract with regards to the data instance.

Ownership can therefore change as a result of:

- a **DDSDataWriter** (p. 1589) in the system with a higher value of the strength that modifies the instance,
- a change in the strength value of the **DDSDataWriter** (p. 1589) that owns the instance, and
- a change in the liveliness of the **DDSDataWriter** (p. 1589) that owns the instance.
- a deadline with regards to the instance that is missed by the **DDSDataWriter** (p. 1589) that owns the instance.

The behavior of the system is as if the determination was made independently by each **DDSDataReader** (p. 1556). Each **DDSDataReader** (p. 1556) may detect the change of ownership at a different time. It is not a requirement that at a particular point in time all the **DDSDataReader** (p. 1556) objects for that **DDSTopic** (p. 1885) have a consistent picture of who owns each instance.

It is also not a requirement that the **DDSDataWriter** (p. 1589) objects are aware of whether they own a particular instance. There is no error or notification given to a **DDSDataWriter** (p. 1589) that modifies an instance it does not currently own.

The requirements are chosen to (a) preserve the decoupling of publishers and subscriber, and (b) allow the policy to be implemented efficiently.

It is possible that multiple **DDSDataWriter** (p. 1589) objects with the same strength modify the same instance. If this occurs RTI Connext will pick one of the **DDSDataWriter** (p. 1589) objects as the owner. It is not specified how the owner is selected. However, the algorithm used to select the owner guarantees that all **DDSDataReader** (p. 1556) objects will make the same choice of the particular **DDSDataWriter** (p. 1589) that is the owner. It also guarantees that the owner remains the same until there is a change in strength, liveliness, the owner misses a deadline on the instance, or a new **DDSDataWriter** (p. 1589) with higher same strength, or a new **DDSDataWriter** (p. 1589) with same strength that should be deemed the owner according to the policy of the Service, modifies the instance.

Exclusive ownership is on an instance-by-instance basis. That is, a subscriber can receive values written by a lower strength **DDSDataWriter** (p. 1589) as long as they affect instances whose values have not been set by the higher-strength **DDSDataWriter** (p. 1589).

9.122.3 Compatibility

The value of the **DDS_OwnershipQosPolicyKind** (p. 612) offered must exactly match the one requested or else they are considered incompatible.

9.122.4 Relationship between registration, liveliness and ownership

The need for registering/unregistering instances stems from two use cases:

- Ownership resolution on redundant systems
- Detection of loss in topological connectivity

These two use cases also illustrate the semantic differences between the **FooDataWriter::unregister_instance** (p. 1948) and **FooDataWriter::dispose** (p. 1957).

9.122.4.1 Ownership Resolution on Redundant Systems

It is expected that users may use DDS to set up redundant systems where multiple **DDSDataWriter** (p. 1589) entities are "capable" of writing the same instance. In this situation, the **DDSDataWriter** (p. 1589) entities are configured such that:

- Either both are writing the instance "constantly"
- Or else they use some mechanism to classify each other as "primary" and "secondary", such that the primary is the only one writing, and the secondary monitors the primary and only writes when it detects that the primary "writer" is no longer writing.

Both cases above use the **DDS_EXCLUSIVE_OWNERSHIP_QOS** (p. 613) and arbitrate themselves by means of the **DDS_OwnershipStrengthQosPolicy** (p. 1227). Regardless of the scheme, the desired behavior from the **DDSDataReader** (p. 1556) point of view is that **DDSDataReader** (p. 1556) normally receives data from the primary unless the "primary" writer stops writing, in which case the **DDSDataReader** (p. 1556) starts to receive data from the secondary **DDSDataWriter** (p. 1589).

This approach requires some mechanism to detect that a **DDSDataWriter** (p. 1589) (the primary) is no longer "writing" the data as it should. There are several reasons why this may happen and all must be detected (but not necessarily distinguished):

crash The writing process is no longer running (e.g. the whole application has crashed)

connectivity loss Connectivity to the writing application has been lost (e.g. network disconnection)

application fault The application logic that was writing the data is faulty and has stopped calling **FooDataWriter::write** (p. 1951).

Arbitrating from a **DDSDataWriter** (p. 1589) to one of a higher strength is simple and the decision can be taken autonomously by the **DDSDataReader** (p. 1556). Switching ownership from a higher strength **DDSDataWriter** (p. 1589) to one of a lower strength **DDSDataWriter** (p. 1589) requires that the **DDSDataReader** (p. 1556) can make a determination that the stronger **DDSDataWriter** (p. 1589) is "no longer writing the instance".

9.122.4.1.1 Case where the data is periodically updated This determination is reasonably simple when the data is being written periodically at some rate. The **DDSDataWriter** (p. 1589) simply states its offered **DDS_DeadlineQos↩ Policy** (p. 928) (maximum interval between updates), and the **DDSDataReader** (p. 1556) automatically monitors that the **DDSDataWriter** (p. 1589) indeed updates the instance at least once per **DDS_DeadlineQosPolicy::period** (p. 930) set by the **DDSDataReader** (p. 1556). If the deadline is missed, the **DDSDataReader** (p. 1556) considers the **DDSData↩ Writer** (p. 1589) "not alive" and automatically gives ownership to the next highest-strength **DDSDataWriter** (p. 1589) that *is* alive.

9.122.4.1.2 Case where data is not periodically updated The case where the **DDSDataWriter** (p. 1589) is not writing data periodically is also a very important use-case. Since the instance is not being updated at any fixed period, the "deadline" mechanism cannot be used to determine ownership. The liveliness solves this situation. Ownership is maintained while the **DDSDataWriter** (p. 1589) is "alive" and for the **DDSDataWriter** (p. 1589) to be alive it must fulfill its **DDS_LivelinessQosPolicy** (p. 1180) contract. The different means to renew liveliness (automatic, manual) combined by the implied renewal each time data is written handle the three conditions above [crash], [connectivity loss], and [application fault]. Note that to handle [application fault], LIVELINESS must be **DDS_MANUAL_BY_TOPIC_LIVELINESS↩ _QOS** (p. 608). The **DDSDataWriter** (p. 1589) can retain ownership by periodically writing data or else calling assert↩ _liveliness if it has no data to write. Alternatively if only protection against [crash] or [connectivity loss] is desired, it is sufficient that some task on the **DDSDataWriter** (p. 1589) process periodically writes data or calls **DDSDomain↩ Participant::assert_liveliness** (p. 1667). However, this scenario requires that the **DDSDataReader** (p. 1556) knows what instances are being "written" by the **DDSDataWriter** (p. 1589). That is the only way that the **DDSDataReader** (p. 1556) deduces the ownership of specific instances from the fact that the **DDSDataWriter** (p. 1589) is still "alive". Hence the need for the **DDSDataWriter** (p. 1589) to "register" and "unregister" instances. Note that while "registration" can be done lazily the first time the **DDSDataWriter** (p. 1589) writes the instance, "unregistration," in general, cannot. Similar reasoning will lead to the fact that unregistration will also require a message to be sent to the **DDSDataReader** (p. 1556).

9.122.4.2 Detection of Loss in Topological Connectivity

There are applications that are designed in such a way that their correct operation requires some minimal topological connectivity, that is, the writer needs to have a minimum number of readers or alternatively the reader must have a minimum number of writers.

A common scenario is that the application does not start doing its logic until it knows that some specific writers have the minimum configured readers (e.g the alarm monitor is up).

A *more* common scenario is that the application logic will wait until some writers appear that can provide some needed source of information (e.g. the raw sensor data that must be processed).

Furthermore, once the application is running it is a requirement that this minimal connectivity (from the source of the data) is monitored and the application informed if it is ever lost. For the case where data is being written periodically, the **DDS_DeadlineQosPolicy** (p. 928) and the on_deadline_missed listener provides the notification. The case where data is not periodically updated requires the use of the **DDS_LivelinessQosPolicy** (p. 1180) in combination with register↩ _instance/unregister_instance to detect whether the "connectivity" has been lost, and the notification is provided by means of **DDS_NOT_ALIVE_NO_WRITERS_INSTANCE_STATE** (p. 166).

In terms of the required mechanisms, the scenario is very similar to the case of maintaining ownership. In both cases, the reader needs to know whether a writer is still "managing the current value of an instance" even though it is not continually writing it and this knowledge requires the writer to keep its liveliness plus some means to know which instances the writer is currently "managing" (i.e. the registered instances).

9.122.4.3 Semantic Difference between unregister_instance and dispose

FooDataWriter::dispose (p. 1957) is semantically different from **FooDataWriter::unregister_instance** (p. 1948). **FooDataWriter::dispose** (p. 1957) indicates that the data instance no longer exists (e.g. a track that has disappeared, a simulation entity that has been destroyed, a record entry that has been deleted, etc.) whereas **FooDataWriter↩ ::unregister_instance** (p. 1948) indicates that the writer is no longer taking responsibility for updating the value of the instance.

Deleting a **DDSDataWriter** (p. 1589) is equivalent to unregistering all the instances it was writing, but is *not* the same as "disposing" all the instances.

For a **DDSTopic** (p. 1885) with **DDS_EXCLUSIVE_OWNERSHIP_QOS** (p. 613), if the current owner of an instance *disposes* it, the readers accessing the instance will see the instance_state as being "DISPOSED" and not see the values being written by the weaker writer (even after the stronger one has disposed the instance). This is because the **DDSDataWriter** (p. 1589) that owns the instance is saying that the instance no longer exists (e.g. the master of the database is saying that a record has been deleted) and thus the readers should see it as such.

For a **DDSTopic** (p. 1885) with **DDS_EXCLUSIVE_OWNERSHIP_QOS** (p. 613), if the current owner of an instance *unregisters* it, then it will relinquish ownership of the instance and thus the readers may see the value updated by another writer (which will then become the owner). This is because the owner said that it no longer will be providing values for the instance and thus another writer can take ownership and provide those values.

9.122.5 Member Data Documentation

9.122.5.1 kind

```
DDS_OwnershipQosPolicyKind DDS_OwnershipQosPolicy::kind
```

<<***basic***>> *(*p. *201)* <<***spec***>> *(*p. *209)* The kind of ownership

[default] DDS_SHARED_OWNERSHIP_QOS (p. 613)

Categories

Immutable (p. 333), **Matching** (p. 382), **Discovery Metadata** (p. 377), **Applied Per-Instance** (p. 375), **Data Availability** (p. 261)

9.123 DDS_OwnershipStrengthQosPolicy Struct Reference

Specifies the value of the strength used to arbitrate among multiple **DDSDataWriter** (p. 1589) objects that attempt to modify the same instance of a data type (identified by **DDSTopic** (p. 1885) + key).

Public Attributes

- **DDS_Long value**

  *<<**basic**>> (p. 201) <<**spec**>> (p. 209) The strength value used to arbitrate among multiple writers.*

9.123.1 Detailed Description

Specifies the value of the strength used to arbitrate among multiple **DDSDataWriter** (p. 1589) objects that attempt to modify the same instance of a data type (identified by **DDSTopic** (p. 1885) + key).

This policy only applies if the **OWNERSHIP** (p. 612) policy is of kind **DDS_EXCLUSIVE_OWNERSHIP_QOS** (p. 613).

Entity:

DDSDataWriter (p. 1589)

Properties:

RxO (p. 556) = N/A
Changeable (p. 556) = **YES** (p. 556)

The value of the **OWNERSHIP_STRENGTH** (p. 613) is used to determine the ownership of a data instance (identified by the key). The arbitration is performed by the **DDSDataReader** (p. 1556).

See also

EXCLUSIVE ownership (p. 1224)

9.123.2 Member Data Documentation

9.123.2.1 value

```
DDS_Long DDS_OwnershipStrengthQosPolicy::value
```

*<<**basic**>> (p. 201) <<**spec**>> (p. 209)* The strength value used to arbitrate among multiple writers.

[default] 0

[range] [0, 1 million]

Categories

Mutable (p. 369), **Discovery Metadata** (p. 377), **Applied Per-Instance** (p. 375), **Data Availability** (p. 261)

9.124 DDS_ParticipantBuiltinTopicData Struct Reference

Entry created when a DomainParticipant object is discovered.

Public Attributes

- **DDS_BuiltinTopicKey_t key**

  DCPS key to distinguish entries.
- struct **DDS_UserDataQosPolicy user_data**

  Policy of the corresponding DomainParticipant.
- struct **DDS_PropertyQosPolicy property**

  *<<**extension**>> (p. 432) Name value pair properties to be stored with DomainParticipant*
- **DDS_ProtocolVersion_t rtps_protocol_version**

  *<<**extension**>> (p. 432) Version number of the RTPS wire protocol used.*
- struct **DDS_VendorId_t rtps_vendor_id**

  *<<**extension**>> (p. 432) ID of vendor implementing the RTPS wire protocol.*
- **DDS_UnsignedLong dds_builtin_endpoints**

  Bitmap of builtin endpoints supported by the participant.
- **DDS_UnsignedLong available_builtin_endpoints_ext**

  Extended bitmap of builtin endpoints supported by the participant.
- struct **DDS_LocatorSeq default_unicast_locators**

  *<<**extension**>> (p. 432) Unicast locators used when individual entities do not specify unicast locators.*
- struct **DDS_ProductVersion_t product_version**

  *<<**extension**>> (p. 432) This is a vendor specific parameter. It gives the current version for rti-dds.*
- struct **DDS_EntityNameQosPolicy participant_name**

  *<<**extension**>> (p. 432) The participant name and role name.*
- **DDS_DomainId_t domain_id**

  *<<**extension**>> (p. 432) Domain ID associated with the discovered participant.*
- struct **DDS_TransportInfoSeq transport_info**

  *<<**extension**>> (p. 432) A sequence of **DDS_TransportInfo_t** (p. 1409) containing information about each of the installed transports of the discovered participant.*
- struct **DDS_Duration_t reachability_lease_duration**

  *<<**extension**>> (p. 432) Locator reachability lease duration.*
- struct **DDS_PartitionQosPolicy partition**

  *<<**extension**>> (p. 432) PartitionQosPolicy of the participant.*
- **DDS_ParticipantTrustProtectionInfo trust_protection_info**

  *<<**extension**>> (p. 432) Trust Plugins protection information associated with the discovered DomainParticipant.*
- **DDS_ParticipantTrustAlgorithmInfo trust_algorithm_info**

  *<<**extension**>> (p. 432) Trust Plugins algorithms associated with the discovered DomainParticipant.*
- **DDS_Boolean partial_configuration**

  *<<**extension**>> (p. 432) Indicates whether a **DDS_ParticipantBuiltinTopicData** (p. 1229) only contains bootstrapping information.*

9.124.1 Detailed Description

Entry created when a DomainParticipant object is discovered.

Data associated with the built-in topic **DDS_PARTICIPANT_TOPIC_NAME** (p. 491). It contains QoS policies and additional information that apply to the remote **DDSDomainParticipant** (p. 1619).

See also

DDS_PARTICIPANT_TOPIC_NAME (p. 491)

DDSParticipantBuiltinTopicDataDataReader (p. 1816)

9.124.2 Member Data Documentation

9.124.2.1 key

DDS_BuiltinTopicKey_t DDS_ParticipantBuiltinTopicData::key

DCPS key to distinguish entries.

9.124.2.2 user_data

struct **DDS_UserDataQosPolicy** DDS_ParticipantBuiltinTopicData::user_data

Policy of the corresponding DomainParticipant.

9.124.2.3 property

struct **DDS_PropertyQosPolicy** DDS_ParticipantBuiltinTopicData::property

<<***extension***>> *(p. 432)* Name value pair properties to be stored with DomainParticipant

9.124.2.4 rtps_protocol_version

DDS_ProtocolVersion_t DDS_ParticipantBuiltinTopicData::rtps_protocol_version

<<***extension***>> *(p. 432)* Version number of the RTPS wire protocol used.

9.124.2.5 rtps_vendor_id

```
struct DDS_VendorId_t DDS_ParticipantBuiltinTopicData::rtps_vendor_id
```

<<***extension***>> *(p. 432)* ID of vendor implementing the RTPS wire protocol.

9.124.2.6 dds_builtin_endpoints

```
DDS_UnsignedLong DDS_ParticipantBuiltinTopicData::dds_builtin_endpoints
```

Bitmap of builtin endpoints supported by the participant.

Each bit indicates a builtin endpoint that may be available on the participant.

Bit	Endpoint
0	SPDPbuiltinParticipantWriter
1	SPDPbuiltinParticipantReader
2	SEDPbuiltinPublicationsWriter
3	SEDPbuiltinPublicationsReader
4	SEDPbuiltinSubscriptionsWriter
5	SEDPbuiltinSubscriptionsReader
10	BuiltinParticipantMessageWriter
11	BuiltinParticipantMessageReader
12	TypeLookupServiceRequestDataWriter
13	TypeLookupServiceRequestDataReader
14	TypeLookupServiceReplyDataWriter
15	TypeLookupServiceReplyDataReader
16	SEDPbuiltinPublicationsSecureWriter
17	SEDPbuiltinPublicationsSecureReader
18	SEDPbuiltinSubscriptionsSecureWriter
19	SEDPbuiltinSubscriptionsSecureReader
20	BuiltinParticipantMessageSecureWriter
21	BuiltinParticipantMessageSecureReader
22	BuiltinParticipantStatelessMessageWriter
23	BuiltinParticipantStatelessMessageReader
24	BuiltinParticipantVolatileMessageSecureWriter
25	BuiltinParticipantVolatileMessageSecureReader
26	SPDPbuiltinParticipantSecureWriter
27	SPDPbuiltinParticipantSecureReader

9.124.2.7 available_builtin_endpoints_ext

DDS_UnsignedLong DDS_ParticipantBuiltinTopicData::available_builtin_endpoints_ext

Extended bitmap of builtin endpoints supported by the participant.

This bitmap is an extension of the **DDS_ParticipantBuiltinTopicData::dds_builtin_endpoints** (p. 1231) field. It is used to indicate additional builtin endpoints that may be available on the participant.

Bit	Endpoint
0	TypeLookupServiceRequestSecureWriter
1	TypeLookupServiceRequestSecureReader
2	TypeLookupServiceReplySecureWriter
3	TypeLookupServiceReplySecureReader

9.124.2.8 default_unicast_locators

struct **DDS_LocatorSeq** DDS_ParticipantBuiltinTopicData::default_unicast_locators

<<***extension***>> *(p. 432)* Unicast locators used when individual entities do not specify unicast locators.

9.124.2.9 product_version

struct **DDS_ProductVersion_t** DDS_ParticipantBuiltinTopicData::product_version

<<***extension***>> *(p. 432)* This is a vendor specific parameter. It gives the current version for rti-dds.

9.124.2.10 participant_name

struct **DDS_EntityNameQosPolicy** DDS_ParticipantBuiltinTopicData::participant_name

<<***extension***>> *(p. 432)* The participant name and role name.

This parameter contains the name and the role name of the discovered participant.

9.124.2.11 domain_id

DDS_DomainId_t DDS_ParticipantBuiltinTopicData::domain_id

<<***extension***>> *(p. 432)* Domain ID associated with the discovered participant.

9.124.2.12 transport_info

```
struct DDS_TransportInfoSeq DDS_ParticipantBuiltinTopicData::transport_info
```

<<***extension***>> *(p. 432)* A sequence of **DDS_TransportInfo_t** (p. 1409) containing information about each of the installed transports of the discovered participant.

This parameter contains a sequence of **DDS_TransportInfo_t** (p. 1409) containing the class_id and message_size_↩ max for all installed transports of the discovered participant. The maximum number of **DDS_TransportInfo_t** (p. 1409) that will be stored in this sequence is controlled by the Domain Participant's resource limit **DDS_DomainParticipant**↩ **ResourceLimitsQosPolicy::transport_info_list_max_length** (p. 1001).

9.124.2.13 reachability_lease_duration

```
struct DDS_Duration_t DDS_ParticipantBuiltinTopicData::reachability_lease_duration
```

<<***extension***>> *(p. 432)* Locator reachability lease duration.

This parameter contains the value of the participant properties: **dds.domain_participant.locator_reachability_**↩ **lease_duration.sec** and **dds.domain_participant.locator_reachability_lease_duration.nanosec** used to configured the locator reachability lease duration.

9.124.2.14 partition

```
struct DDS_PartitionQosPolicy DDS_ParticipantBuiltinTopicData::partition
```

<<***extension***>> *(p. 432)* PartitionQosPolicy of the participant.

9.124.2.15 trust_protection_info

```
DDS_ParticipantTrustProtectionInfo DDS_ParticipantBuiltinTopicData::trust_protection_info
```

<<***extension***>> *(p. 432)* Trust Plugins protection information associated with the discovered DomainParticipant.

Trust Plugins is a generic abstraction that represents any plugin intended to do transformation, interception, and validation of exchanged DomainParticipant data and metadata.

trust_protection_info contains information about how RTPS wire serialization, discovery, and liveliness interact with the loaded Trust Plugins. Two DomainParticipants will not match if their trust_protection_info is incompatible.

The meaning of the contents of this field may vary depending on what Trust Plugins the DomainParticipant is using. For information about how this field interacts with the RTI Security Plugins, please refer to the `RTI Security Plugins User's Manual`.

9.124.2.16 trust_algorithm_info

DDS_ParticipantTrustAlgorithmInfo DDS_ParticipantBuiltinTopicData::trust_algorithm_info

<<***extension***>> *(p. 432)* Trust Plugins algorithms associated with the discovered DomainParticipant.

Trust Plugins is a generic abstraction that represents any plugin intended to do transformation, interception, and validation of exchanged DomainParticipant data and metadata. trust_algorithm_info contains information about what algorithms the loaded Trust Plugins are running. Two DomainParticipants will not match if their trust_algorithm_info are incompatible.

The meaning of the contents of this field may vary depending on what Trust Plugins the DomainParticipant is using. For information about how this field interacts with the RTI Security Plugins, please refer to the `RTI Security Plugins User's Manual`.

9.124.2.17 partial_configuration

DDS_Boolean DDS_ParticipantBuiltinTopicData::partial_configuration

<<***extension***>> *(p. 432)* Indicates whether a **DDS_ParticipantBuiltinTopicData** (p. 1229) only contains bootstrapping information.

If this is **DDS_BOOLEAN_TRUE** (p. 513), the **DDS_ParticipantBuiltinTopicData** (p. 1229) only contains bootstrapping information. If it is **DDS_BOOLEAN_FALSE** (p. 513), it contains both bootstrapping and configuration information. The following fields are valid when this is set to **DDS_BOOLEAN_TRUE** (p. 513):

- **DDS_ParticipantBuiltinTopicData::key** (p. 1230)
- **DDS_ParticipantBuiltinTopicData::property** (p. 1230) (only dds.domain_participant.domain_tag is valid if set)
- **DDS_ParticipantBuiltinTopicData::rtps_protocol_version** (p. 1230)
- **DDS_ParticipantBuiltinTopicData::rtps_vendor_id** (p. 1230)
- **DDS_ParticipantBuiltinTopicData::product_version** (p. 1232)
- **DDS_ParticipantBuiltinTopicData::domain_id** (p. 1232)
- **DDS_ParticipantBuiltinTopicData::transport_info** (p. 1232)
- **DDS_ParticipantBuiltinTopicData::partition** (p. 1233)
- **DDS_ParticipantBuiltinTopicData::trust_protection_info** (p. 1233)
- **DDS_ParticipantBuiltinTopicData::trust_algorithm_info** (p. 1233)

All other fields are invalid.

This field will only be set to **DDS_BOOLEAN_TRUE** (p. 513) if a participant using **DDS_DISCOVERYCONFIG_BUILTIN_SPDP2** (p. 592) receives a bootstrap message and **DDS_DiscoveryConfigQosPolicy::ignore_default_domain_announcements** (p. 950) is set to **DDS_BOOLEAN_FALSE** (p. 513) (non-default).

If a participant is using **DDS_DISCOVERYCONFIG_BUILTIN_SPDP** (p. 591), this field will always be set to **DDS_BOOLEAN_FALSE** (p. 513).

9.125 DDS_ParticipantBuiltinTopicDataSeq Struct Reference

Instantiates **FooSeq** (p. 1965) < **DDS_ParticipantBuiltinTopicData** (p. 1229) > .

9.125.1 Detailed Description

Instantiates **FooSeq** (p. 1965) < **DDS_ParticipantBuiltinTopicData** (p. 1229) > .

Instantiates:

<<*generic*>> *(*p. *433)* **FooSeq** (p. 1965)

See also

DDS_ParticipantBuiltinTopicData (p. 1229)

9.126 DDS_ParticipantTrustAlgorithmInfo Struct Reference

Trust Plugins algorithm information associated with the discovered DomainParticipant.

Public Attributes

- **DDS_ParticipantTrustSignatureAlgorithmInfo signature**

  Information regarding algorithms for validation of data and metadata exchanged by the DomainParticipant.
- **DDS_ParticipantTrustKeyEstablishmentAlgorithmInfo key_establishment**

  Information regarding algorithms for transformation of data and metadata exchanged between two DomainParticipants.
- **DDS_ParticipantTrustInterceptorAlgorithmInfo interceptor**

  Information regarding algorithms for interception of data and metadata exchanged by the DomainParticipant.

9.126.1 Detailed Description

Trust Plugins algorithm information associated with the discovered DomainParticipant.

9.126.2 Member Data Documentation

9.126.2.1 signature

DDS_ParticipantTrustSignatureAlgorithmInfo DDS_ParticipantTrustAlgorithmInfo::signature

Information regarding algorithms for validation of data and metadata exchanged by the DomainParticipant.

9.126.2.2 key_establishment

DDS_ParticipantTrustKeyEstablishmentAlgorithmInfo DDS_ParticipantTrustAlgorithmInfo::key_establishment

Information regarding algorithms for transformation of data and metadata exchanged between two DomainParticipants.

9.126.2.3 interceptor

DDS_ParticipantTrustInterceptorAlgorithmInfo DDS_ParticipantTrustAlgorithmInfo::interceptor

Information regarding algorithms for interception of data and metadata exchanged by the DomainParticipant.

9.127 DDS_ParticipantTrustInterceptorAlgorithmInfo Struct Reference

Trust Plugins interception algorithm information associated with the discovered DomainParticipant.

Public Attributes

- **DDS_TrustAlgorithmSet supported_mask**

  Trust Plugins algorithms supported for interception of data and metadata exchanged by the DomainParticipant.
- **DDS_TrustAlgorithmSet builtin_endpoints_required_mask**

  Trust Plugins algorithms used for interception of metadata exchanged by the discovery, service request, and liveliness builtin endpoints.
- **DDS_TrustAlgorithmSet builtin_kx_endpoints_required_mask**

  Trust Plugins algorithms used for interception of metadata exchanged by the key exchange builtin endpoints.
- **DDS_TrustAlgorithmSet user_endpoints_default_required_mask**

  Default trust plugin algorithms used for interception of data exchanged by the user endpoints. This is the algorithm that remote DomainParticipants should assume when ***DDS_EndpointTrustInterceptorAlgorithmInfo*** *(*p. 1144*) is not serialized.*

9.127.1 Detailed Description

Trust Plugins interception algorithm information associated with the discovered DomainParticipant.

9.127.2 Member Data Documentation

9.127.2.1 supported_mask

DDS_TrustAlgorithmSet DDS_ParticipantTrustInterceptorAlgorithmInfo::supported_mask

Trust Plugins algorithms supported for interception of data and metadata exchanged by the DomainParticipant.

9.127.2.2 builtin_endpoints_required_mask

DDS_TrustAlgorithmSet DDS_ParticipantTrustInterceptorAlgorithmInfo::builtin_endpoints_required_↩
mask

Trust Plugins algorithms used for interception of metadata exchanged by the discovery, service request, and liveliness builtin endpoints.

9.127.2.3 builtin_kx_endpoints_required_mask

DDS_TrustAlgorithmSet DDS_ParticipantTrustInterceptorAlgorithmInfo::builtin_kx_endpoints_required↩
_mask

Trust Plugins algorithms used for interception of metadata exchanged by the key exchange builtin endpoints.

9.127.2.4 user_endpoints_default_required_mask

DDS_TrustAlgorithmSet DDS_ParticipantTrustInterceptorAlgorithmInfo::user_endpoints_default_↩
required_mask

Default trust plugin algorithms used for interception of data exchanged by the user endpoints. This is the algorithm that remote DomainParticipants should assume when **DDS_EndpointTrustInterceptorAlgorithmInfo** (p. 1144) is not serialized.

9.128 DDS_ParticipantTrustKeyEstablishmentAlgorithmInfo Struct Reference

Trust Plugins key establishment algorithm information associated with the discovered DomainParticipant.

Public Attributes

- **DDS_TrustAlgorithmRequirements shared_secret**

  Trust Plugins key establishment algorithm requirements of the DomainParticipant in the context of deriving a shared secret.

9.128.1 Detailed Description

Trust Plugins key establishment algorithm information associated with the discovered DomainParticipant.

9.128.2 Member Data Documentation

9.128.2.1 shared_secret

DDS_TrustAlgorithmRequirements DDS_ParticipantTrustKeyEstablishmentAlgorithmInfo::shared_secret

Trust Plugins key establishment algorithm requirements of the DomainParticipant in the context of deriving a shared secret.

9.129 DDS_ParticipantTrustProtectionInfo Struct Reference

Trust Plugins Protection information associated with the discovered DomainParticipant.

Public Attributes

- **DDS_ParticipantTrustAttributesMask bitmask**

  Information about how RTPS wire serialization, discovery, and liveliness interact with the associated Trust Plugins Protection configuration.
- **DDS_PluginParticipantTrustAttributesMask plugin_bitmask**

  Internal plugin information that is opaque to DDS.

9.129.1 Detailed Description

Trust Plugins Protection information associated with the discovered DomainParticipant.

9.129.2 Member Data Documentation

9.129.2.1 bitmask

DDS_ParticipantTrustAttributesMask DDS_ParticipantTrustProtectionInfo::bitmask

Information about how RTPS wire serialization, discovery, and liveliness interact with the associated Trust Plugins Protection configuration.

The meaning of the contents of this field may vary depending on what Trust Plugins the DomainParticipant is using. For information about how this field interacts with the RTI Security Plugins, please refer to the RTI Security Plugins User's Manual.

9.129.2.2 plugin_bitmask

DDS_PluginParticipantTrustAttributesMask DDS_ParticipantTrustProtectionInfo::plugin_bitmask

Internal plugin information that is opaque to DDS.

The meaning of the contents of this field may vary depending on what Trust Plugins the DomainParticipant is using. For information about how this field interacts with the RTI Security Plugins, please refer to the RTI Security Plugins User's Manual.

9.130 DDS_ParticipantTrustSignatureAlgorithmInfo Struct Reference

Trust Plugins signature algorithm information associated with the discovered DomainParticipant.

Public Attributes

- **DDS_TrustAlgorithmRequirements trust_chain**

  Trust Plugins signature algorithm requirements of the DomainParticipant in the context of creating a chain of trust.
- **DDS_TrustAlgorithmRequirements message_auth**

  Trust Plugins signature algorithm requirements of the DomainParticipant in the context of checking that messages are authentic.

9.130.1 Detailed Description

Trust Plugins signature algorithm information associated with the discovered DomainParticipant.

9.130.2 Member Data Documentation

9.130.2.1 trust_chain

DDS_TrustAlgorithmRequirements DDS_ParticipantTrustSignatureAlgorithmInfo::trust_chain

Trust Plugins signature algorithm requirements of the DomainParticipant in the context of creating a chain of trust.

9.130.2.2 message_auth

DDS_TrustAlgorithmRequirements DDS_ParticipantTrustSignatureAlgorithmInfo::message_auth

Trust Plugins signature algorithm requirements of the DomainParticipant in the context of checking that messages are authentic.

9.131 DDS_PartitionQosPolicy Struct Reference

<<***spec***>> *(*p. *209)* Set of strings that introduces logical partitions in **DDSDomainParticipant** (p. 1619), **DDSPublisher** (p. 1818), or **DDSSubscriber** (p. 1860) entities.

Public Attributes

- struct **DDS_StringSeq name**
  <<***basic***>> (p. *201)* <<***spec***>> (p. *209) A list of partition names*

9.131.1 Detailed Description

<<***spec***>> *(*p. *209)* Set of strings that introduces logical partitions in **DDSDomainParticipant** (p. 1619), **DDSPublisher** (p. 1818), or **DDSSubscriber** (p. 1860) entities.

This QoS policy is used to set string identifiers that are used for partitioning entities that would otherwise be connected to and exchange data with each other:

- A **DDSDataWriter** (p. 1589) within a **DDSPublisher** (p. 1818) only communicates with a **DDSDataReader** (p. 1556) in a **DDSSubscriber** (p. 1860) if (in addition to matching the **DDSTopic** (p. 1885) and having compatible QoS) the **DDSPublisher** (p. 1818) and **DDSSubscriber** (p. 1860) have a common partition name string.
- **DDSDomainParticipant** (p. 1619) entities (with the same domain ID and domain tag) are visible to each other only if they have at least one partition name string in common.

Entity:

DDSPublisher (p. 1818), **DDSSubscriber** (p. 1860), **DDSDomainParticipant** (p. 1619)

Properties:

RxO (p. 556) = NO
Changeable (p. 556) = **YES** (p. 556)

9.131.2 Usage

The Partition QoS policy provides another way to control which entities will match-and thus communicate with-which other entities. It can be used to prevent entities that would have otherwise matched from talking to each other. Much in the same way that only applications within the same DDS domain will communicate with each other, only entities that belong to the same partition can talk to each other.

The Partition QoS policy allows you to add one or more strings, "partitions", to an entity:

- A DataWriter and DataReader for the same topic are only considered matched if their Publishers and Subscribers have partitions in common (intersecting partitions).
- DomainParticipants (with the same domain ID and domain tag) are visible to each other only if they have at least one partition in common.

Since the set of partitions for an entity can be dynamically changed, the Partition QoS policy is useful for creating temporary separation groups among entities that would otherwise be connected to and exchange data with each other.

DomainParticipant partitions and Publisher/Subscriber partitions are independent of each other. You can use both features independently or in combination to provide the right level of isolation.

Failure to match partitions is not considered an incompatible QoS and does not trigger any listeners or conditions. A change in this policy *can* potentially modify the "match" of existing DataReader and DataWriter entities. It may establish new "matches" that did not exist before, or break existing matches.

Partition strings are usually directly matched via string comparisons. However, partition strings can also contain wildcard symbols so that partitions can be matched via pattern matching. As long as the partitions or wildcard patterns of an entity intersect with the partitions or wildcard patterns of otherwise matching entities, the entities match; otherwise they do not.

These partition name patterns are regular expressions as defined by the POSIX fnmatch API (1003.2-1992 section B.6). A **DDSDomainParticipant** (p. 1619), **DDSPublisher** (p. 1818), or **DDSSubscriber** (p. 1860) entity may include regular expressions in partition names, but no two names that both contain wildcards will ever be considered to match. This means that although regular expressions may be used on the entities, RTI Connext will not try to match two regular expressions.

Each entity must belong to at least one logical partition. A regular expression is not considered to be a logical partition. If an entity has not specified a logical partition, it is assumed to be in the default partition. The default partition is defined to be an empty string (""). Put another way:

- An empty sequence of strings in this QoS policy is considered equivalent to a sequence containing only a single string, the empty string.
- A string sequence that contains only regular expressions and no literal strings, it is treated as if it had an additional element, the empty string.

Partitions are different from creating **DDSEntity** (p. 1731) objects in different domains in several ways.

- First, entities belonging to different domains are completely isolated from each other; there is no traffic, meta-traffic or any other way for an application or RTI Connext itself to see entities in a domain it does not belong to.
- Second, a **DDSEntity** (p. 1731) can only belong to one domain whereas a **DDSEntity** (p. 1731) can be in multiple partitions.
- Finally, as far as RTI Connext is concerned, each unique data instance is identified by the tuple **(DomainID, domain tag, DDSTopic (**p. **1885), key)**. Therefore two **DDSEntity** (p. 1731) objects in different domains cannot refer to the same data instance. On the other hand, the same data instance can be made available (published) or requested (subscribed) on one or more partitions.

For more information, see the "PARTITION QosPolicy" section of the `Core Libraries User's Manual`.

9.131.3 Member Data Documentation

9.131.3.1 name

```
struct DDS_StringSeq DDS_PartitionQosPolicy::name
```

<<***basic***>> *(p. 201)* <<***spec***>> *(p. 209)* A list of partition names

Several restrictions apply to the partition names in this sequence. A violation of one of the following rules will result in a **DDS_RETCODE_INCONSISTENT_POLICY** (p. 533) when setting a **DDSPublisher** (p. 1818)'s or **DDSSubscriber** (p. 1860)'s QoS.

- A partition name string cannot be NULL, nor can it contain the reserved comma character (',').
- The maximum number of partition name strings allowable in a **DDS_PartitionQosPolicy** (p. 1240) is specified on a domain basis in **DDS_DomainParticipantResourceLimitsQosPolicy::max_partitions** (p. 993). The length of this sequence may not be greater than that value.
- The maximum cumulative length of all partition name strings in a **DDS_PartitionQosPolicy** (p. 1240) is specified on a domain basis in **DDS_DomainParticipantResourceLimitsQosPolicy::max_partition_cumulative_↩ characters** (p. 993).

The memory for the strings in this sequence is managed according to the conventions described in **String Conventions** (p. 758). In particular, be careful to avoid a situation in which RTI Connext allocates a string on your behalf and you then reuse that string in such a way that RTI Connext believes it to have more memory allocated to it than it actually does.

[default] Empty sequence (zero-length sequence). Since no logical partition is specified, RTI Connext will assume the entity to be in default partition (empty string partition "").

[range] List of partition name with above restrictions

Categories

Mutable (p. 369), **Discovery Isolation** (p. 277), **Matching** (p. 382), **Discovery Metadata** (p. 377)

9.132 DDS_PersistentStorageSettings Struct Reference

Configures durable writer history and durable reader state.

Public Attributes

- **DDS_Boolean enable**

  <<***extension***>> (p. 213) *Enables durable writer history in a* ***DDSDataWriter*** (p. 1589) *and durable reader state in a* ***DDSDataReader*** (p. 1556).
- char * **file_name**

  <<***extension***>> (p. 213) *The file name where the durable writer history or durable reader state will be stored.*
- char * **trace_file_name**

  <<***extension***>> (p. 213) *The file name where to store the SQL statements executed when loading and storing the durable writer history or durable reader state.*
- **DDS_PersistentJournalKind journal_kind**

  <<***extension***>> (p. 213) *Sets the journal mode of the persistent storage.*
- **DDS_PersistentSynchronizationKind synchronization_kind**

  <<***extension***>> (p. 213) *Sets the level of synchronization with the physical disk.*
- **DDS_Boolean vacuum**

  <<***extension***>> (p. 213) *Sets the auto-vacuum status of the storage.*
- **DDS_Boolean restore**

  <<***extension***>> (p. 213) *Indicates if the persisted writer history or reader state must be restored.*
- struct **DDS_AllocationSettings_t writer_instance_cache_allocation**

  <<***extension***>> (p. 213) *Configures the resource limits associated with the instance durable writer history cache.*
- struct **DDS_AllocationSettings_t writer_sample_cache_allocation**

  <<***extension***>> (p. 213) *Configures the resource limits associated with the sample durable writer history cache.*
- **DDS_Boolean writer_memory_state**

  <<***extension***>> (p. 213) *Determines how much state will be kept in memory by the durable writer history in order to avoid accessing the persistent storage in disk.*
- **DDS_UnsignedLong reader_checkpoint_frequency**

  <<***extension***>> (p. 213) *Controls how often the reader state is stored into the database.*

9.132.1 Detailed Description

Configures durable writer history and durable reader state.

In a **DDSDataWriter** (p. 1589), this structure configures durable writer history. This feature allows a DataWriter to persist its historical cache, so that it can survive shutdowns, crashes, and restarts. When an application restarts, each DataWriter that has been configured to have durable writer history automatically loads all of the data in this cache from disk and can carry on sending data as if it had never stopped executing.

In a **DDSDataReader** (p. 1556), this structure configures durable reader state. This feature allows a DataReader to persist its state and remember which data it has already received. When an application restarts, each DataReader that has been configured to have durable reader state automatically loads its state from disk and can carry on receiving data as if it had never stopped executing. Data that had already been received by the DataReader before the restart will be suppressed so that it is not even sent over the network.

RTI Connext uses SQLite to store the durable writer history and durable reader state.

QoS:

DDS_DurabilityQosPolicy (p. 1005)

9.132.2 Member Data Documentation

9.132.2.1 enable

```
DDS_Boolean DDS_PersistentStorageSettings::enable
```

<<***extension***>> *(p. 213)* Enables durable writer history in a **DDSDataWriter** (p. 1589) and durable reader state in a **DDSDataReader** (p. 1556).

When this field is set to **DDS_BOOLEAN_TRUE** (p. 513), the persistent storage configuration using this structure will take precedence over the configuration using the deprecated **dds.data_writer.history.odbc_plugin.builtin**.∗ and **dds.data_reader.state**.∗ properties.

[default] DDS_BOOLEAN_FALSE (p. 513)

Categories

Immutable (p. 333), **Durability** (p. 278)

9.132.2.2 file_name

```
char* DDS_PersistentStorageSettings::file_name
```

<<***extension***>> *(p. 213)* The file name where the durable writer history or durable reader state will be stored.

Setting this field to a value other than NULL is mandatory when enabling durable writer history or durable reader state.

If the file does not exist, it will be created.

If the file exists and **DDS_PersistentStorageSettings::restore** (p. 1246) is set to **DDS_BOOLEAN_TRUE** (p. 513), the durable writer history or durable reader state will be restored from the file. Otherwise, the file will be overwritten.

Important: When the file exists, the fields **DDS_DataReaderProtocolQosPolicy::virtual_guid** (p. 844) and **DDS_DataWriterProtocolQosPolicy::virtual_guid** (p. 893) will be set by RTI Connext based on the file content. If you change these fields, the value will be ignored.

RTI Connext uses SQLite to store the durable writer history and durable reader state.

[default] NULL

Categories

Immutable (p. 333), **Durability** (p. 278)

9.132.2.3 trace_file_name

```
char* DDS_PersistentStorageSettings::trace_file_name
```

<<***extension***>> *(p. 213)* The file name where to store the SQL statements executed when loading and storing the durable writer history or durable reader state.

Setting this field to a value other than NULL will enable tracing of the SQL statements executed when loading and storing the durable writer history or durable reader state.

Important: Enabling tracing will have a negative impact on performance. Use this feature only for debugging purposes.

[default] NULL

Categories

Immutable (p. 333), **Durability** (p. 278)

9.132.2.4 journal_kind

```
DDS_PersistentJournalKind DDS_PersistentStorageSettings::journal_kind
```

<<***extension***>> *(p. 213)* Sets the journal mode of the persistent storage.

[default] DDS_WAL_PERSISTENT_JOURNAL (p. 597)

Categories

Immutable (p. 333), **Durability** (p. 278)

9.132.2.5 synchronization_kind

```
DDS_PersistentSynchronizationKind DDS_PersistentStorageSettings::synchronization_kind
```

<<***extension***>> *(p. 213)* Sets the level of synchronization with the physical disk.

[default] DDS_NORMAL_PERSISTENT_SYNCHRONIZATION (p. 598)

Categories

Immutable (p. 333), **Durability** (p. 278)

9.132.2.6 vacuum

```
DDS_Boolean DDS_PersistentStorageSettings::vacuum
```

<<***extension***>> *(*p. *213)* Sets the auto-vacuum status of the storage.

When auto-vacuum is **DDS_BOOLEAN_TRUE** (p. 513), the storage files will be compacted automatically with every transaction.

When auto-vacuum is **DDS_BOOLEAN_FALSE** (p. 513), after data is deleted from the storage files, the files remain the same size.

[default] DDS_BOOLEAN_TRUE (p. 513)

Categories

Immutable (p. 333), **Durability** (p. 278)

9.132.2.7 restore

```
DDS_Boolean DDS_PersistentStorageSettings::restore
```

<<***extension***>> *(*p. *213)* Indicates if the persisted writer history or reader state must be restored.

For a **DDSDataWriter** (p. 1589), this field indicates whether or not the persisted writer history must be restored once the DataWriter is restarted.

For a **DDSDataReader** (p. 1556), this field indicates whether or not the persisted reader state must be restored once the DataReader is restarted.

[default] DDS_BOOLEAN_TRUE (p. 513)

Categories

Immutable (p. 333), **Durability** (p. 278)

9.132.2.8 writer_instance_cache_allocation

`struct` **`DDS_AllocationSettings_t`** `DDS_PersistentStorageSettings::writer_instance_cache_allocation`

<<***extension***>> *(p. 213)* Configures the resource limits associated with the instance durable writer history cache.

This field only applies to **DDSDataWriter** (p. 1589) entities.

To minimize the number of accesses to the persisted storage, RTI Connext uses an instance cache.

Do not confuse this limit with the initial and maximum number of instances that can be maintained by a DataWriter in persistent storage. These resource limits are configured using **DDS_ResourceLimitsQosPolicy::max_instances** (p. 1311) and **DDS_ResourceLimitsQosPolicy::initial_instances** (p. 1312).

If **DDS_PersistentStorageSettings::writer_memory_state** (p. 1248) is set to **DDS_BOOLEAN_TRUE** (p. 513), then the value of **DDS_AllocationSettings_t::max_count** (p. 797) is set to **DDS_LENGTH_UNLIMITED** (p. 636), overwriting any value set by the user.

DDS_AllocationSettings_t::incremental_count (p. 797) is ignored.

[range] DDS_AllocationSettings_t::max_count (p. 797) in interval [1, INT_MAX], **DDS_LENGTH_AUTO** (p. 631), or **DDS_LENGTH_UNLIMITED** (p. 636). **DDS_AllocationSettings_t::initial_count** (p. 797) in interval [1, INT_MAX], or **DDS_LENGTH_AUTO** (p. 631).

DDS_LENGTH_AUTO (p. 631) means that the value will be set to the equivalent value of **DDS_ResourceLimitsQos↩ Policy** (p. 1308).

[default] DDS_AllocationSettings_t::max_count (p. 797) = **DDS_LENGTH_AUTO** (p. 631) (= **DDS_Resource↩ LimitsQosPolicy::max_instances** (p. 1311)) and **DDS_AllocationSettings_t::initial_count** (p. 797) = **DDS_↩ LENGTH_AUTO** (p. 631) (= **DDS_ResourceLimitsQosPolicy::initial_instances** (p. 1312)).

Categories

Immutable (p. 333), **Durability** (p. 278), **Sample Memory Management** (p. 321)

9.132.2.9 writer_sample_cache_allocation

`struct` **`DDS_AllocationSettings_t`** `DDS_PersistentStorageSettings::writer_sample_cache_allocation`

<<***extension***>> *(p. 213)* Configures the resource limits associated with the sample durable writer history cache.

This field only applies to **DDSDataWriter** (p. 1589) entities.

To minimize the number of accesses to the persisted storage, RTI Connext uses a sample cache.

Do not confuse this limit with the initial and maximum number of samples that can be maintained by a DataWriter in persistent storage. These resource limits are configured using **DDS_ResourceLimitsQosPolicy::max_samples** (p. 1311) and **DDS_ResourceLimitsQosPolicy::initial_samples** (p. 1312).

DDS_AllocationSettings_t::incremental_count (p. 797) is ignored.

[range] DDS_AllocationSettings_t::max_count (p. 797) in interval [1, INT_MAX], **DDS_LENGTH_AUTO** (p. 631), or **DDS_LENGTH_UNLIMITED** (p. 636). **DDS_AllocationSettings_t::initial_count** (p. 797) in interval [1, INT_MAX], or **DDS_LENGTH_AUTO** (p. 631).

DDS_LENGTH_AUTO (p. 631) means that the value will be set to the equivalent value of **DDS_ResourceLimitsQos↩ Policy** (p. 1308).

[default] DDS_AllocationSettings_t::max_count (p. 797) = 32 and **DDS_AllocationSettings_t::initial_count** (p. 797) = 32.

Categories

Immutable (p. 333), **Durability** (p. 278), **Sample Memory Management** (p. 321)

9.132.2.10 writer_memory_state

`DDS_Boolean DDS_PersistentStorageSettings::writer_memory_state`

<<***extension***>> *(p. 213)* Determines how much state will be kept in memory by the durable writer history in order to avoid accessing the persistent storage in disk.

This field only applies to **DDSDataWriter** (p. 1589) entities.

If this field is set to **DDS_BOOLEAN_TRUE** (p. 513), then **DDS_AllocationSettings_t::max_count** (p. 797) in **DDS_PersistentStorageSettings::writer_instance_cache_allocation** (p. 1246) is set to **DDS_LENGTH_UNLIMITED** (p. 636), overwriting any value set by the user.

In addition, the durable writer history will keep a fixed state overhead per sample in memory. This mode provides the best durable writer history performance. However, the restore operation will be slower, and the maximum number of samples that the durable writer history can manage is limited by the available physical memory.

If this field is set to **DDS_BOOLEAN_FALSE** (p. 513), all the state will be kept in the underlying database. In this mode, the maximum number of samples in the durable writer history is not limited by the physical memory available.

This field is always set to **DDS_BOOLEAN_FALSE** (p. 513) when the DataWriter is configured with **DDS_ReliabilityQosPolicy::acknowledgment_kind** (p. 1298) set to **DDS_APPLICATION_AUTO_ACKNOWLEDGMENT_MODE** (p. 633) or **DDS_APPLICATION_EXPLICIT_ACKNOWLEDGMENT_MODE** (p. 634), or **DDS_AvailabilityQosPolicy::enable_required_subscriptions** (p. 808) is set to **DDS_BOOLEAN_TRUE** (p. 513).

[default] DDS_BOOLEAN_TRUE (p. 513)

Categories

Immutable (p. 333), **Durability** (p. 278), **Middleware Memory Management** (p. 297)

9.132.2.11 reader_checkpoint_frequency

`DDS_UnsignedLong DDS_PersistentStorageSettings::reader_checkpoint_frequency`

<<***extension***>> *(p. 213)* Controls how often the reader state is stored into the database.

This field only applies to **DDSDataReader** (p. 1556) entities.

A value of N means store the state once every N received and processed samples.

The circumstances under which a data sample is considered "processed by the application" depends on the DataReader configuration.

For additional information on when a sample is considered "processed by the application" see `Durable Reader State`, in the Core Libraries User's Manual.

A high value will provide better performance. However, if the DataReader is restarted it may receive some duplicate samples.

[range] [1, 1000000] **[default]** 1

Categories

Immutable (p. 333), **Durability** (p. 278)

9.133 DDS_PresentationQosPolicy Struct Reference

<<***spec***>> *(*p. *209)* Specifies how the samples representing changes to data instances are presented to a subscribing application.

Public Attributes

- **DDS_PresentationQosPolicyAccessScopeKind access_scope**

  <<***spec***>> *(*p. *209) Determines the largest scope spanning the entities for which the order and coherency of changes can be preserved.*
- **DDS_Boolean coherent_access**

  <<***spec***>> *(*p. *209) Specifies support for coherent access. Controls whether coherent access is supported within the scope* `access_scope`*.*
- **DDS_Boolean ordered_access**

  <<***spec***>> *(*p. *209) Specifies support for ordered access to the samples received at the subscription end. Controls whether ordered access is supported within the scope* `access_scope`*.*
- **DDS_Boolean drop_incomplete_coherent_set**

  <<***extension***>> *(*p. *213) Indicates whether or not a* ***DDSDataReader*** *(*p. *1556) should drop samples from an incomplete coherent set (one for which not all the samples were received). Such samples are reported as lost in the SAMPLE_LOST Status.*

9.133.1 Detailed Description

<<***spec***>> *(*p. *209)* Specifies how the samples representing changes to data instances are presented to a subscribing application.

This QoS policy controls the extent to which changes to data instances can be made dependent on each other and also the kind of dependencies that can be propagated and maintained by RTI Connext. Specifically, this policy affects the application's ability to:

- specify and receive coherent changes to instances
- specify the relative order in which changes are presented

Entity:

DDSPublisher (p. 1818), **DDSSubscriber** (p. 1860)

Status:

DDS_OFFERED_INCOMPATIBLE_QOS_STATUS (p. 540), **DDS_REQUESTED_INCOMPATIBLE_QOS_↩ STATUS** (p. 540)

Properties:

RxO (p. 556) = YES
Changeable (p. 556) = **UNTIL ENABLE** (p. 556)

9.133.2 Usage

A **DDSDataReader** (p. 1556) will usually receive data in the order that it was sent by a **DDSDataWriter** (p. 1589), and the data is presented to the **DDSDataReader** (p. 1556) as soon as the application receives the next expected value. However, sometimes you may want a set of data for the same DataWriter or different DataWriters to be presented to the DataReader(s) only after *all* of the elements of the set have been received, but not before. Or you may want the data to be presented in a different order than that in which it was sent. Specifically for keyed data, you may want the middleware to present the data in keyed – or *instance* – order, such that samples pertaining to the same instance are presented together.

The Presentation QoS policy is a request-offered QoS policy that allows you to specify different *scopes* of presentation. It also controls whether or not a set of changes within the scope is delivered at the same time or can be delivered as soon as each element is received.

- `coherent_access` controls whether RTI Connext will preserve the groupings of changes made by a publishing application by means of the operations **DDSPublisher::begin_coherent_changes** (p. 1832) and **DDSPublisher::end_coherent_changes** (p. 1833).

- `ordered_access` controls whether RTI Connext will preserve the order of changes.

- `access_scope` controls the granularity of the other settings. See below:

If `coherent_access` is set, then the `access_scope` set on the **DDSSubscriber** (p. 1860) controls the maximum extent of coherent changes. The behavior is as follows:

- If `access_scope` is set to **DDS_INSTANCE_PRESENTATION_QOS** (p. 616) (the default), the behavior is the same as **DDS_TOPIC_PRESENTATION_QOS** (p. 616).

- If `access_scope` is set to **DDS_TOPIC_PRESENTATION_QOS** (p. 616), then coherent changes (indicated by their enclosure within calls to **DDSPublisher::begin_coherent_changes** (p. 1832) and **DDSPublisher::end_↩ coherent_changes** (p. 1833)) will be made available as a unit to each remote **DDSDataReader** (p. 1556) independently. That is, changes made to instances within each individual **DDSDataWriter** (p. 1589) will be presented as a unit. They will not be grouped with changes made to instances belonging to a different **DDSDataWriter** (p. 1589).

- If access_scope is set to **DDS_GROUP_PRESENTATION_QOS** (p. 616), then coherent changes made to instances through a set of **DDSDataWriter** (p. 1589) entities attached to a common **DDSPublisher** (p. 1818) will be made available as a unit to the **DDSDataReader** (p. 1556) entities within a **DDSSubscriber** (p. 1860).

If `ordered_access` is set, then the `access_scope` set on the **DDSSubscriber** (p. 1860) controls the maximum extent to which order will be preserved by RTI Connext.

- If `access_scope` is set to **DDS_INSTANCE_PRESENTATION_QOS** (p. 616) (the lowest level), then the relative order of DDS samples sent by a **DDSDataWriter** (p. 1589) is only preserved on a per-instance basis. If two DDS samples refer to the same instance (identified by Topic and a particular value for the key), then the order in which they are stored in the **DDSDataReader** (p. 1556) queue is consistent with the order in which the changes occurred. However, if the two DDS samples belong to different instances, the order in which they are presented may or may not match the order in which the changes occurred. This is the case even if it is the same application thread making the changes using the same **DDSDataWriter** (p. 1589).

- If access_scope is set to **DDS_TOPIC_PRESENTATION_QOS** (p. 616), changes (creations, deletions, modifications) made by a single **DDSDataWriter** (p. 1589) to one or more instances are presented to a **DDSDataReader** (p. 1556) in the same order in which they occur. Changes made to instances though different **DDSDataWriter** (p. 1589) entities are not required to be presented in the order in which they occur. This is the case even if the changes are made by a single application thread using **DDSDataWriter** (p. 1589) objects attached to the same **DDSPublisher** (p. 1818).

- Finally, if access_scope is set to **DDS_GROUP_PRESENTATION_QOS** (p. 616), changes made to instances via **DDSDataWriter** (p. 1589) entities attached to the same **DDSPublisher** (p. 1818) object are presented to the **DDSDataReader** (p. 1556) entities within a **DDSSubscriber** (p. 1860) in the same order in which they occur. For this to happen, the application must take the samples using the Subscriber's **DDSSubscriber::begin_access** (p. 1873) and **DDSSubscriber::end_access** (p. 1874) operations.

See the "Ordered Access" section in the "PRESENTATION QosPolicy" section of the Core Libraries User's Manual for more information.

If access_scope is set to **DDS_HIGHEST_OFFERED_PRESENTATION_QOS** (p. 616) on the **DDSSubscriber** (p. 1860), then the **DDSSubscriber** (p. 1860) will use the access_scope offered by the remote **DDSPublisher** (p. 1818).

Note that this QoS policy controls the scope at which related changes are made available to the subscriber. This means the subscriber **can** access the changes in a coherent manner and in the proper order; however, it does not necessarily imply that the **DDSSubscriber** (p. 1860) **will** indeed access the changes in the correct order. For that to occur, the application at the subscriber end must use the proper logic in reading the **DDSDataReader** (p. 1556) objects.

For **DDS_GROUP_PRESENTATION_QOS** (p. 616) the subscribing application must use the APIs **DDSSubscriber::begin_access** (p. 1873), **DDSSubscriber::end_access** (p. 1874) and **DDSSubscriber::get_datareaders** (p. 1874) to access the changes in the proper order. If you do not use these operations, the data may not be ordered across DataWriters that belong to the same **DDSPublisher** (p. 1818).

The field **DDS_SampleInfo::coherent_set_info** (p. 1354) is set when a sample is part of a coherent set. This field provides information to identify the coherent set that a sample is part of. In addition, **DDS_CoherentSetInfo_t::incomplete_coherent_set** (p. 826) indicates if a sample is part of an incomplete coherent set (one for which not all samples have been received). Coherent sets for which some of the samples are filtered out by content or time on the **DDSDataWriter** (p. 1589) are considered incomplete.

By default, the samples that are received from an incomplete coherent set are dropped by the DataReader(s) and they are not provided to the application. Such samples are reported as lost in the SAMPLE_LOST Status. By setting **DDS_PresentationQosPolicy::drop_incomplete_coherent_set** (p. 1253) to **DDS_BOOLEAN_FALSE** (p. 513), you can change this behavior and, in this case, samples from incomplete coherent sets will be provided to the application. These samples have **DDS_CoherentSetInfo_t::incomplete_coherent_set** (p. 826) set to **DDS_BOOLEAN_TRUE** (p. 513).

9.133.3 Compatibility

The value offered is considered compatible with the value requested if and only if the following conditions are met:

- the inequality *offered* access_scope >= *requested* access_scope evaluates to 'TRUE' or requested access_scope is **DDS_HIGHEST_OFFERED_PRESENTATION_QOS** (p. 616). For the purposes of this inequality, the values of access_scope are considered ordered such that **DDS_INSTANCE_PRESENTATION_QOS** (p. 616) < **DDS_TOPIC_PRESENTATION_QOS** (p. 616) < **DDS_GROUP_PRESENTATION_QOS** (p. 616).

- requested `coherent_access` is **DDS_BOOLEAN_FALSE** (p. 513), or else both offered and requested `coherent_access` are **DDS_BOOLEAN_TRUE** (p. 513).

- requested `ordered_access` is **DDS_BOOLEAN_FALSE** (p. 513), or else both offered and requested `ordered_access` are **DDS_BOOLEAN_TRUE** (p. 513).

9.133.4 Member Data Documentation

9.133.4.1 access_scope

DDS_PresentationQosPolicyAccessScopeKind DDS_PresentationQosPolicy::access_scope

<<***spec***>> *(p. 209)* Determines the largest scope spanning the entities for which the order and coherency of changes can be preserved.

[default] DDS_INSTANCE_PRESENTATION_QOS (p. 616)

Categories

Immutable (p. 333), **Matching** (p. 382), **Discovery Metadata** (p. 377), **Applied Per-Instance** (p. 375), **Sample Ordering** (p. 324), **Data Availability** (p. 261)

9.133.4.2 coherent_access

DDS_Boolean DDS_PresentationQosPolicy::coherent_access

<<***spec***>> *(p. 209)* Specifies support for *coherent* access. Controls whether coherent access is supported within the scope `access_scope`.

That is, the ability to group a set of changes as a unit on the publishing end such that they are received as a unit at the subscribing end.

Note: To use this feature, the DataWriter must be configured for RELIABLE communication (see **DDS_RELIABLE_↩ RELIABILITY_QOS** (p. 633)).

[default] DDS_BOOLEAN_FALSE (p. 513)

Categories

Immutable (p. 333), **Matching** (p. 382), **Discovery Metadata** (p. 377), **Sample Ordering** (p. 324), **Data Availability** (p. 261)

9.133.4.3 ordered_access

```
DDS_Boolean DDS_PresentationQosPolicy::ordered_access
```

<<***spec***>> *(p. 209)* Specifies support for *ordered* access to the samples received at the subscription end. Controls whether ordered access is supported within the scope `access_scope`.

That is, the ability of the subscriber to see changes in the same order as they occurred on the publishing end.

[default] DDS_BOOLEAN_FALSE (p. 513)

Categories

Immutable (p. 333), **Matching** (p. 382), **Discovery Metadata** (p. 377), **Applied Per-Instance** (p. 375), **Sample Ordering** (p. 324), **Data Availability** (p. 261)

9.133.4.4 drop_incomplete_coherent_set

```
DDS_Boolean DDS_PresentationQosPolicy::drop_incomplete_coherent_set
```

<<***extension***>> *(p. 213)* Indicates whether or not a **DDSDataReader** (p. 1556) should drop samples from an incomplete coherent set (one for which not all the samples were received). Such samples are reported as lost in the SAMPLE_LOST Status.

Note that a coherent set will be considered incomplete if some of its samples are filtered by content or time on the DataWriter side.

By default, the samples that are received from an incomplete coherent set are dropped (and reported as lost) by the DataReader(s) and they are not provided to the application. By setting this parameter to **DDS_BOOLEAN_FALSE** (p. 513), you can change this behavior.

Samples from an incomplete coherent set have **DDS_CoherentSetInfo_t::incomplete_coherent_set** (p. 826) set to **DDS_BOOLEAN_TRUE** (p. 513) in **DDS_SampleInfo::coherent_set_info** (p. 1354).

[default] DDS_BOOLEAN_TRUE (p. 513)

Categories

Immutable (p. 333), **Sample Ordering** (p. 324), **Data Availability** (p. 261)

9.134 DDS_PrintFormatProperty Struct Reference

A collection of attributes used to configure how data samples will be formatted when converted to a string.

Public Attributes

- **DDS_PrintFormatKind kind**

  The kind of format to be used when converting a data sample to a string.
- **DDS_Boolean pretty_print**

  Choose whether to print a data sample in a more readable format or to eliminate all white space.
- **DDS_Boolean enum_as_int**

  Choose whether to print enum values as integers or as strings.
- **DDS_Boolean include_root_elements**

  Choose whether or not to include the root elements of the print format in the output string.

9.134.1 Detailed Description

A collection of attributes used to configure how data samples will be formatted when converted to a string.

Examples

HelloWorldPlugin.cxx.

9.134.2 Member Data Documentation

9.134.2.1 kind

```
DDS_PrintFormatKind DDS_PrintFormatProperty::kind
```

The kind of format to be used when converting a data sample to a string.

Data samples can be represented in a default, XML, or JSON format.

See also

DDS_PrintFormatKind (p. 66)

[default] DDS_DEFAULT_PRINT_FORMAT (p. 67)

9.134.2.2 pretty_print

```
DDS_Boolean DDS_PrintFormatProperty::pretty_print
```

Choose whether to print a data sample in a more readable format or to eliminate all white space.

A value of **DDS_BOOLEAN_TRUE** (p. 513) will add indentation and newlines to the string representation of the data sample making it more readable. For example:

```
<FooType>
   <myLong>5</myLong>
</FooType>
```

A value of **DDS_BOOLEAN_FALSE** (p. 513) will cause all white space in the string representation of the data sample to be eliminated. For example:

```
<FooType><myLong>5</myLong></FooType>
```

[default] DDS_BOOLEAN_TRUE (p. 513)

9.134.2.3 enum_as_int

```
DDS_Boolean DDS_PrintFormatProperty::enum_as_int
```

Choose whether to print enum values as integers or as strings.

A value of **DDS_BOOLEAN_TRUE** (p. 513) will cause enumeration literals to be printed using their integer value. For example:

```
<FooType>
   <myEnum>5</myEnum>
</FooType>
```

A value of **DDS_BOOLEAN_FALSE** (p. 513) will cause enumeration literals to be printed as strings using their member names. For example:

```
<FooType>
   <myEnum>RED</myEnum>
</FooType>
```

[default] DDS_BOOLEAN_FALSE (p. 513)

9.134.2.4 include_root_elements

```
DDS_Boolean DDS_PrintFormatProperty::include_root_elements
```

Choose whether or not to include the root elements of the print format in the output string.

This field only applies if the **DDS_PrintFormatProperty::kind** (p. 1254) is **DDS_XML_PRINT_FORMAT** (p. 67) or **DDS_JSON_PRINT_FORMAT** (p. 67)

If the value is **DDS_BOOLEAN_TRUE** (p. 513) and the kind is **DDS_XML_PRINT_FORMAT** (p. 67) then a top-level tag with the type's name in it will be included in the output. For example:

```
<FooType>
   <myLong>5</myLong>
</FooType>
```

If the value is **DDS_BOOLEAN_TRUE** (p. 513) and the kind is **DDS_JSON_PRINT_FORMAT** (p. 67) then top-level opening and closing braces will be included in the output. For example:

```
{
   "myLong":5
}
```

If the value is **DDS_BOOLEAN_FALSE** (p. 513) the elements described above will not be included in the output. For example:

```
<myLong>5</myLong>
"myLong":5
```

[default] DDS_BOOLEAN_TRUE (p. 513)

9.135 DDS_ProductVersion_t Struct Reference

*<<**extension**>> (p. 432)* Type used to represent the current version of RTI Connext.

Public Attributes

- **DDS_Char major**

  Major product version.
- **DDS_Char minor**

  Minor product version.
- **DDS_Char release**

  Release letter for product version.
- **DDS_Char revision**

  Revision number of product.

9.135.1 Detailed Description

<<***extension***>> *(*p. *432)* Type used to represent the current version of RTI Connext.

9.135.2 Member Data Documentation

9.135.2.1 major

```
DDS_Char DDS_ProductVersion_t::major
```

Major product version.

9.135.2.2 minor

```
DDS_Char DDS_ProductVersion_t::minor
```

Minor product version.

9.135.2.3 release

```
DDS_Char DDS_ProductVersion_t::release
```

Release letter for product version.

9.135.2.4 revision

```
DDS_Char DDS_ProductVersion_t::revision
```

Revision number of product.

9.136 DDS_ProfileQosPolicy Struct Reference

<<***extension***>> *(p. 213)* Configures the way that XML documents containing QoS profiles are loaded by RTI Connext.

Public Attributes

- struct **DDS_StringSeq string_profile**

  <<***basic***>> *(p. 201)* <<***extension***>> *(p. 213) Sequence of strings containing a XML document to load.*
- struct **DDS_StringSeq url_profile**

  <<***basic***>> *(p. 201)* <<***extension***>> *(p. 213) Sequence of* ***URL groups*** *(p. 392) containing a set of XML documents to load.*
- **DDS_Boolean ignore_user_profile**

  <<***basic***>> *(p. 201)* <<***extension***>> *(p. 213) Ignores the file USER_QOS_PROFILES.xml in the current working directory.*
- **DDS_Boolean ignore_environment_profile**

  <<***basic***>> *(p. 201)* <<***extension***>> *(p. 213) Ignores the value of the* ***NDDS_QOS_PROFILES*** *(p. 392).*
- **DDS_Boolean ignore_resource_profile**

  <<***basic***>> *(p. 201)* <<***extension***>> *(p. 213) Ignores the file NDDS_QOS_PROFILES.xml.*

9.136.1 Detailed Description

<<***extension***>> *(p. 213)* Configures the way that XML documents containing QoS profiles are loaded by RTI Connext.

All QoS values for Entities can be configured in QoS profiles defined in XML documents. XML documents can be passed to RTI Connext in string form or, more likely, through files found on a file system.

There are also default locations where DomainParticipants will look for files to load QoS profiles. These include the current working directory from where an application is started, a file in the distribution directory for RTI Connext, and the locations specified by an environment variable.

You may disable any or all of these default locations using the Profile QoS policy.

Entity:

DDSDomainParticipantFactory (p. 1694)

Properties:

RxO (p. 556) = NO
Changeable (p. 556) = **Changeable** (p. 556)

9.136.2 Member Data Documentation

9.136.2.1 string_profile

```
struct DDS_StringSeq DDS_ProfileQosPolicy::string_profile
```

<<***basic***>> *(p. 201)* <<***extension***>> *(p. 213)* Sequence of strings containing a XML document to load.

The concatenation of the strings in this sequence must be a valid XML document according to the XML QoS profile schema.

[default] Empty sequence (zero-length).

Categories

Immutable (p. 333), **QoS Profile Management** (p. 312)

9.136.2.2 url_profile

```
struct DDS_StringSeq DDS_ProfileQosPolicy::url_profile
```

<<***basic***>> *(p. 201)* <<***extension***>> *(p. 213)* Sequence of **URL groups** (p. 392) containing a set of XML documents to load.

Only one of the elements of each group will be loaded by RTI Connext, starting from the left.

[default] Empty sequence (zero-length).

Categories

Immutable (p. 333), **QoS Profile Management** (p. 312)

9.136.2.3 ignore_user_profile

```
DDS_Boolean DDS_ProfileQosPolicy::ignore_user_profile
```

<<***basic***>> *(p. 201)* <<***extension***>> *(p. 213)* Ignores the file USER_QOS_PROFILES.xml in the current working directory.

When this field is set to **DDS_BOOLEAN_TRUE** (p. 513), the QoS profiles contained in the file USER_QOS_↩ PROFILES.xml in the current working directory will be ignored.

[default] DDS_BOOLEAN_FALSE (p. 513)

Categories

Immutable (p. 333), **QoS Profile Management** (p. 312)

9.136.2.4 ignore_environment_profile

```
DDS_Boolean DDS_ProfileQosPolicy::ignore_environment_profile
```

<<***basic***>> (p. *201)* <<***extension***>> (p. *213)* Ignores the value of the **NDDS_QOS_PROFILES** (p. 392).

When this field is set to **DDS_BOOLEAN_TRUE** (p. 513), the value of the environment variable NDDS_QOS_PROFILES will be ignored.

[default] DDS_BOOLEAN_FALSE (p. 513)

Categories

Immutable (p. 333), **QoS Profile Management** (p. 312)

9.136.2.5 ignore_resource_profile

```
DDS_Boolean DDS_ProfileQosPolicy::ignore_resource_profile
```

<<***basic***>> (p. *201)* <<***extension***>> (p. *213)* Ignores the file NDDS_QOS_PROFILES.xml.

When this field is set to **DDS_BOOLEAN_TRUE** (p. 513), the QoS profiles contained in the file NDDS_QOS_↩ PROFILES.xml in $NDDSHOME/resource/xml will be ignored.

[default] DDS_BOOLEAN_FALSE (p. 513)

Categories

Immutable (p. 333), **QoS Profile Management** (p. 312)

9.137 DDS_Property_t Struct Reference

Properties are name/value pairs objects.

Public Attributes

- char * **name**

  Property name.
- char * **value**

  Property value.
- **DDS_Boolean propagate**

  Indicates if the property must be propagated on discovery.

9.137.1 Detailed Description

Properties are name/value pairs objects.

9.137.2 Member Data Documentation

9.137.2.1 name

```
char* DDS_Property_t::name
```

Property name.

It must be a NULL-terminated string allocated with **DDS_String_alloc** (p. 759) or **DDS_String_dup** (p. 761).

9.137.2.2 value

```
char* DDS_Property_t::value
```

Property value.

It must be a NULL-terminated string allocated with **DDS_String_alloc** (p. 759) or **DDS_String_dup** (p. 761).

9.137.2.3 propagate

```
DDS_Boolean DDS_Property_t::propagate
```

Indicates if the property must be propagated on discovery.

9.138 DDS_PropertyQosPolicy Struct Reference

<<***extension***>> *(p. 213)* Stores name/value(string) pairs that can be used to configure certain parameters of RTI Connext that are not exposed through formal QoS policies. Can also be used to store and propagate application-specific name/value pairs that can be retrieved by user code during discovery.

Public Attributes

- struct **DDS_PropertySeq value**

  <<***basic***>> *(p. 201)* <<***extension***>> *(p. 213) Sequence of properties.*

9.138.1 Detailed Description

<<***extension***>> *(p. 213)* Stores name/value(string) pairs that can be used to configure certain parameters of RTI Connext that are not exposed through formal QoS policies. Can also be used to store and propagate application-specific name/value pairs that can be retrieved by user code during discovery.

Entity:

DDSDomainParticipant (p. 1619) **DDSDataReader** (p. 1556) **DDSDataWriter** (p. 1589)

Properties:

RxO (p. 556) = N/A;
Changeable (p. 556) = **YES** (p. 556)

See also

DDSDomainParticipant::get_builtin_subscriber (p. 1661)

9.138.2 Usage

The PROPERTY QoS policy can be used to associate a set of properties in the form of (name, value) pairs with a **DDSDataReader** (p. 1556), **DDSDataWriter** (p. 1589), or **DDSDomainParticipant** (p. 1619). This is similar to the **DDS_UserDataQosPolicy** (p. 1504), except this policy uses (name, value) pairs, and you can select whether or not a particular pair should be propagated (included in the builtin topic).

You can find a complete list of predefined properties in the `Property Reference Guide`.

This QoS policy may be used to configure:

- Durable Writer History, see **Configuring Durable Writer History** (p. 198)
- Durable Reader State, see **Configuring Durable Reader State** (p. 200)
- Builtin Transport Plugins, see **UDPv4 Transport Property Names in Property QoS Policy of Domain Participant** (p. 465), **UDPv6 Transport Property Names in Property QoS Policy of Domain Participant** (p. 481), and **Shared Memory Transport Property Names in Property QoS Policy of Domain Participant** (p. 459)
- Extension Transport Plugins, see **Loading Transport Plugins through Property QoS Policy of Domain Participant** (p. 178)
- **Clock Selection** (p. 40)

In addition, you may add your own name/value pairs to the Property QoS policy of an Entity. Via this QoS policy, you can direct RTI Connext to propagate these name/value pairs with the discovery information for the Entity. Applications that discover the Entity can then access the user-specific name/value pairs in the discovery information of the remote Entity. This allows you to add meta-information about an Entity for application-specific use, for example, authentication/authorization certificates (which can also be done using the **DDS_UserDataQosPolicy** (p. 1504) or **DDS_Group↩ DataQosPolicy** (p. 1160)).

9.138.2.1 Reasons for Using the PropertyQosPolicy

- Supports dynamic loading of extension transports
- Supports multiple instances of the builtin transports
- Allows full pluggable transport configuration for non-C/C++ language bindings (Java, .NET, etc.)
- Avoids the process of creating entities disabled, changing their QoS settings, then enabling them
- Allows selection of clock

Some of the RTI Connext capabilities configurable via the Property QoS policy can also be configured in code via APIs. However, the Property QoS policy allows you to configure those parameters via XML files. In addition, some of the configuration APIs will only work if the Entity was created in a disabled state and then enabled after the configuration change was applied. By configuring those parameters using the Property QoS policy during entity creation, you avoid the additional work of first creating a disabled entity and then enabling it afterwards.

There are helper functions to facilitate working with properties, see the **PROPERTY** (p. 617) page.

9.138.3 Member Data Documentation

9.138.3.1 value

`struct` **`DDS_PropertySeq`** `DDS_PropertyQosPolicy::value`

*<<**basic**>> (p. 201) <<**extension**>> (p. 213)* Sequence of properties.

[default] An empty list.

Categories

Mutable (p. 369), **Discovery Metadata** (p. 377), **Application-Defined Metadata** (p. 254)

9.139 DDS_PropertySeq Struct Reference

Declares IDL `sequence` < **DDS_Property_t** (p. 1259) >

9.139.1 Detailed Description

Declares IDL `sequence` < **DDS_Property_t** (p. 1259) >

See also

DDS_Property_t (p. 1259)

9.140 DDS_ProtocolVersion_t Struct Reference

<<***extension***>> *(p. 432)* Type used to represent the version of the RTPS protocol.

Public Attributes

- **DDS_Octet major**
  Major protocol version number.
- **DDS_Octet minor**
  Minor protocol version number.

9.140.1 Detailed Description

<<***extension***>> *(p. 432)* Type used to represent the version of the RTPS protocol.

9.140.2 Member Data Documentation

9.140.2.1 major

```
DDS_Octet DDS_ProtocolVersion_t::major
```

Major protocol version number.

9.140.2.2 minor

```
DDS_Octet DDS_ProtocolVersion_t::minor
```

Minor protocol version number.

9.141 DDS_PublicationBuiltinTopicData Struct Reference

Entry created when a **DDSDataWriter** (p. 1589) is discovered in association with its Publisher.

Public Attributes

- **DDS_BuiltinTopicKey_t key**

  DCPS key to distinguish entries.
- **DDS_BuiltinTopicKey_t participant_key**

  DCPS key of the participant to which the DataWriter belongs.
- char * **topic_name**

  *Name of the related **DDSTopic** (p. 1885).*
- char * **type_name**

  *Name of the type attached to the **DDSTopic** (p. 1885).*
- struct **DDS_DurabilityQosPolicy durability**

  durability policy of the corresponding DataWriter
- struct **DDS_DurabilityServiceQosPolicy durability_service**

  durability_service policy of the corresponding DataWriter
- struct **DDS_DeadlineQosPolicy deadline**

  Policy of the corresponding DataWriter.
- struct **DDS_LatencyBudgetQosPolicy latency_budget**

  Policy of the corresponding DataWriter.
- struct **DDS_LivelinessQosPolicy liveliness**

  Policy of the corresponding DataWriter.
- struct **DDS_ReliabilityQosPolicy reliability**

  Policy of the corresponding DataWriter.
- struct **DDS_LifespanQosPolicy lifespan**

  Policy of the corresponding DataWriter.
- struct **DDS_UserDataQosPolicy user_data**

  Policy of the corresponding DataWriter.
- struct **DDS_OwnershipQosPolicy ownership**

  Policy of the corresponding DataWriter.
- struct **DDS_OwnershipStrengthQosPolicy ownership_strength**

  Policy of the corresponding DataWriter.
- struct **DDS_DestinationOrderQosPolicy destination_order**

  Policy of the corresponding DataWriter.
- struct **DDS_PresentationQosPolicy presentation**

  Policy of the Publisher to which the DataWriter belongs.
- struct **DDS_PartitionQosPolicy partition**

  Policy of the Publisher to which the DataWriter belongs.
- struct **DDS_TopicDataQosPolicy topic_data**

  Policy of the related Topic.
- struct **DDS_GroupDataQosPolicy group_data**

  Policy of the Publisher to which the DataWriter belongs.
- struct **DDS_DataRepresentationQosPolicy representation**

  Data representation policy of the corresponding DataWriter.

- **DDS_DataTagQosPolicy data_tags**

  Tags of the corresponding DataWriter.
- struct **DDS_TypeCode** * **type_code**

  <<***extension***>> *(p. 432) Type code information of the corresponding Topic.*
- **DDS_BuiltinTopicKey_t publisher_key**

  <<***extension***>> *(p. 432) DCPS key of the publisher to which the DataWriter belongs*
- struct **DDS_PropertyQosPolicy property**

  <<***extension***>> *(p. 432) Properties of the corresponding DataWriter.*
- struct **DDS_LocatorSeq unicast_locators**

  <<***extension***>> *(p. 432) Custom unicast locators that the endpoint can specify. The default locators will be used if this is not specified.*
- struct **DDS_GUID_t virtual_guid**

  <<***extension***>> *(p. 432) Virtual GUID associated to the DataWriter.*
- struct **DDS_ServiceQosPolicy service**

  <<***extension***>> *(p. 432) Policy of the corresponding DataWriter.*
- **DDS_ProtocolVersion_t rtps_protocol_version**

  <<***extension***>> *(p. 432) Version number of the RTPS wire protocol used.*
- struct **DDS_VendorId_t rtps_vendor_id**

  <<***extension***>> *(p. 432) ID of vendor implementing the RTPS wire protocol.*
- struct **DDS_ProductVersion_t product_version**

  <<***extension***>> *(p. 432) This is a vendor specific parameter. It gives the current version for rti-dds.*
- struct **DDS_LocatorFilterQosPolicy locator_filter**

  <<***extension***>> *(p. 432) Policy of the corresponding DataWriter*
- **DDS_Boolean disable_positive_acks**

  <<***extension***>> *(p. 432) This is a vendor specific parameter. Determines whether matching DataReaders send positive acknowledgements for reliability.*
- struct **DDS_EntityNameQosPolicy publication_name**

  <<***extension***>> *(p. 432) The publication name and role name.*
- struct **DDS_EndpointTrustProtectionInfo trust_protection_info**

  <<***extension***>> *(p. 432) Trust Plugins protection information associated with the discovered DataWriter.*
- struct **DDS_EndpointTrustAlgorithmInfo trust_algorithm_info**

  <<***extension***>> *(p. 432) Trust Plugins algorithms associated with the discovered DataWriter.*
- **DDS_BuiltinTopicKey_t related_datareader_key**

  <<***extension***>> *(p. 432) Key of the related DataReader that is created as a part of the Remote Procedure Call (RPC) or Request-Reply endpoints.*

9.141.1 Detailed Description

Entry created when a **DDSDataWriter** (p. 1589) is discovered in association with its Publisher.

Data associated with the built-in topic **DDS_PUBLICATION_TOPIC_NAME** (p. 494). It contains QoS policies and additional information that apply to the remote **DDSDataWriter** (p. 1589) the related **DDSPublisher** (p. 1818).

See also

DDS_PUBLICATION_TOPIC_NAME (p. 494)

DDSPublicationBuiltinTopicDataDataReader (p. 1817)

9.141.2 Member Data Documentation

9.141.2.1 key

```
DDS_BuiltinTopicKey_t DDS_PublicationBuiltinTopicData::key
```

DCPS key to distinguish entries.

9.141.2.2 participant_key

```
DDS_BuiltinTopicKey_t DDS_PublicationBuiltinTopicData::participant_key
```

DCPS key of the participant to which the DataWriter belongs.

9.141.2.3 topic_name

```
char* DDS_PublicationBuiltinTopicData::topic_name
```

Name of the related **DDSTopic** (p. 1885).

The length of this string is limited to 255 characters.

The memory for this field is managed as described in **String Conventions** (p. 758).

See also

String Conventions (p. 758)

9.141.2.4 type_name

```
char* DDS_PublicationBuiltinTopicData::type_name
```

Name of the type attached to the **DDSTopic** (p. 1885).

The length of this string is limited to 255 characters.

The memory for this field is managed as described in **String Conventions** (p. 758).

See also

String Conventions (p. 758)

9.141.2.5 durability

`struct` **DDS_DurabilityQosPolicy** DDS_PublicationBuiltinTopicData::durability

durability policy of the corresponding DataWriter

9.141.2.6 durability_service

`struct` **DDS_DurabilityServiceQosPolicy** DDS_PublicationBuiltinTopicData::durability_service

durability_service policy of the corresponding DataWriter

9.141.2.7 deadline

`struct` **DDS_DeadlineQosPolicy** DDS_PublicationBuiltinTopicData::deadline

Policy of the corresponding DataWriter.

9.141.2.8 latency_budget

`struct` **DDS_LatencyBudgetQosPolicy** DDS_PublicationBuiltinTopicData::latency_budget

Policy of the corresponding DataWriter.

9.141.2.9 liveliness

`struct` **DDS_LivelinessQosPolicy** DDS_PublicationBuiltinTopicData::liveliness

Policy of the corresponding DataWriter.

9.141.2.10 reliability

`struct` **DDS_ReliabilityQosPolicy** DDS_PublicationBuiltinTopicData::reliability

Policy of the corresponding DataWriter.

9.141.2.11 lifespan

`struct` **DDS_LifespanQosPolicy** DDS_PublicationBuiltinTopicData::lifespan

Policy of the corresponding DataWriter.

9.141.2.12 user_data

`struct` **DDS_UserDataQosPolicy** DDS_PublicationBuiltinTopicData::user_data

Policy of the corresponding DataWriter.

9.141.2.13 ownership

`struct` **DDS_OwnershipQosPolicy** DDS_PublicationBuiltinTopicData::ownership

Policy of the corresponding DataWriter.

9.141.2.14 ownership_strength

`struct` **DDS_OwnershipStrengthQosPolicy** DDS_PublicationBuiltinTopicData::ownership_strength

Policy of the corresponding DataWriter.

9.141.2.15 destination_order

`struct` **DDS_DestinationOrderQosPolicy** DDS_PublicationBuiltinTopicData::destination_order

Policy of the corresponding DataWriter.

Warning

Only the field **DDS_DestinationOrderQosPolicy::kind** (p. 933) is propagated during discovery. The other fields always contain their default values.

9.141.2.16 presentation

struct **DDS_PresentationQosPolicy** DDS_PublicationBuiltinTopicData::presentation

Policy of the Publisher to which the DataWriter belongs.

9.141.2.17 partition

struct **DDS_PartitionQosPolicy** DDS_PublicationBuiltinTopicData::partition

Policy of the Publisher to which the DataWriter belongs.

9.141.2.18 topic_data

struct **DDS_TopicDataQosPolicy** DDS_PublicationBuiltinTopicData::topic_data

Policy of the related Topic.

9.141.2.19 group_data

struct **DDS_GroupDataQosPolicy** DDS_PublicationBuiltinTopicData::group_data

Policy of the Publisher to which the DataWriter belongs.

9.141.2.20 representation

struct **DDS_DataRepresentationQosPolicy** DDS_PublicationBuiltinTopicData::representation

Data representation policy of the corresponding DataWriter.

9.141.2.21 data_tags

DDS_DataTagQosPolicy DDS_PublicationBuiltinTopicData::data_tags

Tags of the corresponding DataWriter.

9.141.2.22 type_code

```
struct DDS_TypeCode* DDS_PublicationBuiltinTopicData::type_code
```

<<***extension***>> *(p. 432)* Type code information of the corresponding Topic.

If using TypeObject v2 (**DDS_DISCOVERYCONFIG_TYPE_LOOKUP_SERVICE_CHANNEL** (p. 592)), which is the default, this is NULL if the type is not yet resolved. If the remote endpoint's topic name matches the **DDS_Discovery↩ ConfigQosPolicy::request_types_filter** (p. 959), this will always contain the TypeCode converted from the complete TypeObject v2 representation (if non-NULL); the minimal TypeObject v2 will not be provided even if available. If the topic name does not match the **DDS_DiscoveryConfigQosPolicy::request_types_filter** (p. 959) and the complete Type↩ Object v2 is resolved, this will contain the TypeCode converted from the complete TypeObject v2 representation. If the topic name does not match the **DDS_DiscoveryConfigQosPolicy::request_types_filter** (p. 959), and only the minimal TypeObject v2 is resolved, this will contain the TypeCode converted from the minimal TypeObject v2 representation. As such, information that is not available in the minimal TypeObject v2 representation will be lost (such as default values) and member names will be the stringified version of the hashed member name, rather than the full member name. See `TypeLookup Service` for more information.

9.141.2.23 publisher_key

```
DDS_BuiltinTopicKey_t DDS_PublicationBuiltinTopicData::publisher_key
```

<<***extension***>> *(p. 432)* DCPS key of the publisher to which the DataWriter belongs

9.141.2.24 property

```
struct DDS_PropertyQosPolicy DDS_PublicationBuiltinTopicData::property
```

<<***extension***>> *(p. 432)* Properties of the corresponding DataWriter.

9.141.2.25 unicast_locators

```
struct DDS_LocatorSeq DDS_PublicationBuiltinTopicData::unicast_locators
```

<<***extension***>> *(p. 432)* Custom unicast locators that the endpoint can specify. The default locators will be used if this is not specified.

9.141.2.26 virtual_guid

```
struct DDS_GUID_t DDS_PublicationBuiltinTopicData::virtual_guid
```

<<***extension***>> *(p. 432)* Virtual GUID associated to the DataWriter.

See also

DDS_GUID_t (p. 1161)

9.141.2.27 service

```
struct DDS_ServiceQosPolicy DDS_PublicationBuiltinTopicData::service
```

<<***extension***>> *(p. 432)* Policy of the corresponding DataWriter.

9.141.2.28 rtps_protocol_version

```
DDS_ProtocolVersion_t DDS_PublicationBuiltinTopicData::rtps_protocol_version
```

<<***extension***>> *(p. 432)* Version number of the RTPS wire protocol used.

9.141.2.29 rtps_vendor_id

```
struct DDS_VendorId_t DDS_PublicationBuiltinTopicData::rtps_vendor_id
```

<<***extension***>> *(p. 432)* ID of vendor implementing the RTPS wire protocol.

9.141.2.30 product_version

```
struct DDS_ProductVersion_t DDS_PublicationBuiltinTopicData::product_version
```

<<***extension***>> *(p. 432)* This is a vendor specific parameter. It gives the current version for rti-dds.

9.141.2.31 locator_filter

struct **DDS_LocatorFilterQosPolicy** DDS_PublicationBuiltinTopicData::locator_filter

<<***extension***>> *(p. 432)* Policy of the corresponding DataWriter

Related to **DDS_MultiChannelQosPolicy** (p. 1214).

9.141.2.32 disable_positive_acks

DDS_Boolean DDS_PublicationBuiltinTopicData::disable_positive_acks

<<***extension***>> *(p. 432)* This is a vendor specific parameter. Determines whether matching DataReaders send positive acknowledgements for reliability.

9.141.2.33 publication_name

struct **DDS_EntityNameQosPolicy** DDS_PublicationBuiltinTopicData::publication_name

<<***extension***>> *(p. 432)* The publication name and role name.

This member contains the name and the role name of the discovered publication.

9.141.2.34 trust_protection_info

struct **DDS_EndpointTrustProtectionInfo** DDS_PublicationBuiltinTopicData::trust_protection_info

<<***extension***>> *(p. 432)* Trust Plugins protection information associated with the discovered DataWriter.

Trust Plugins is a generic abstraction that represents any plugin intended to do transformation, interception, and validation of exchanged data and metadata.

trust_protection_info contains information about how RTPS wire serialization, discovery, and liveliness interact with the loaded Trust Plugins. Two endpoints will not match if their trust_protection_info is incompatible.

The meaning of the contents of this field may vary depending on what Trust Plugins the endpoint is using. For information about how this field interacts with the RTI Security Plugins, please refer to the `RTI Security Plugins User's Manual`.

9.141.2.35 trust_algorithm_info

```
struct DDS_EndpointTrustAlgorithmInfo DDS_PublicationBuiltinTopicData::trust_algorithm_info
```

<<***extension***>> *(p. 432)* Trust Plugins algorithms associated with the discovered DataWriter.

Trust Plugins is a generic abstraction that represents any plugin intended to do transformation, interception, and validation of exchanged data and metadata.

trust_algorithm_info contains information about what algorithms the loaded Trust Plugins are running. Two endpoints will not match if their trust_algorithm_info are incompatible.

The meaning of the contents of this field may vary depending on what Trust Plugins the endpoint is using. For information about how this field interacts with the RTI Security Plugins, please refer to the `RTI Security Plugins User's Manual`.

9.141.2.36 related_datareader_key

```
DDS_BuiltinTopicKey_t DDS_PublicationBuiltinTopicData::related_datareader_key
```

<<***extension***>> *(p. 432)* Key of the related DataReader that is created as a part of the Remote Procedure Call (RPC) or Request-Reply endpoints.

This field will be set to something other than **DDS_BUILTIN_TOPIC_KEY_UNKNOWN** (p. 508) only when the discovered DataWriter is a part of Remote Procedure Call (RPC) or Request-Reply endpoints.

9.142 DDS_PublicationBuiltinTopicDataSeq Struct Reference

Instantiates **FooSeq** (p. 1965) < **DDS_PublicationBuiltinTopicData** (p. 1264) > .

9.142.1 Detailed Description

Instantiates **FooSeq** (p. 1965) < **DDS_PublicationBuiltinTopicData** (p. 1264) > .

Instantiates:

<<***generic***>> *(p. 433)* **FooSeq** (p. 1965)

See also

DDS_PublicationBuiltinTopicData (p. 1264)

9.143 DDS_PublicationMatchedStatus Struct Reference

DDS_PUBLICATION_MATCHED_STATUS (p. 543)

Public Attributes

- **DDS_Long total_count**

  *The total cumulative number of times that this **DDSDataWriter** (p. 1589) discovered a "match" with a **DDSDataReader** (p. 1556).*
- **DDS_Long total_count_change**

  The changes in total_count since the last time the listener was called or the status was read.
- **DDS_Long current_count**

  *The current number of DataReaders with which this **DDSDataWriter** (p. 1589) is matched.*
- **DDS_Long current_count_peak**

  *<<**extension**>> (p. 432) Greatest number of DataReaders that matched this **DDSDataWriter** (p. 1589) simultaneously.*
- **DDS_Long current_count_change**

  The change in current_count since the last time the listener was called or the status was read.
- **DDS_InstanceHandle_t last_subscription_handle**

  *This InstanceHandle can be used to look up which remote **DDSDataReader** (p. 1556) was the last to cause this DataWriter's status to change, using **DDSDataWriter::get_matched_subscription_data** (p. 1600).*

9.143.1 Detailed Description

DDS_PUBLICATION_MATCHED_STATUS (p. 543)

A "match" happens when the **DDSDataWriter** (p. 1589) finds a **DDSDataReader** (p. 1556) with the same **DDSTopic** (p. 1885), same or compatible data type, and requested QoS that is compatible with that offered by the **DDSDataWriter** (p. 1589). (For information on compatible data types, see the `Extensible Types Guide`.)

This status is also changed (and the listener, if any, called) when a match is ended. A local **DDSDataWriter** (p. 1589) will become "unmatched" from a remote **DDSDataReader** (p. 1556) when that **DDSDataReader** (p. 1556) goes away for any of the following reasons:

- The matched **DDSDataReader** (p. 1556)'s **DDSDomainParticipant** (p. 1619) has lost liveliness.
- This DataWriter or the matched DataReader has changed QoS such that the entities are now incompatible.
- The matched DataReader has been deleted.

This status may reflect changes from multiple match or unmatch events, and the **DDS_PublicationMatchedStatus::current_count_change** (p. 1275) can be used to determine the number of changes since the listener was called back or the status was checked.

9.143.2 Member Data Documentation

9.143.2.1 total_count

DDS_Long DDS_PublicationMatchedStatus::total_count

The total cumulative number of times that this **DDSDataWriter** (p. 1589) discovered a "match" with a **DDSDataReader** (p. 1556).

This number increases whenever a new match is discovered. It does not decrease when an existing match goes away for any of the reasons described in **DDS_PublicationMatchedStatus** (p. 1273).

9.143.2.2 total_count_change

DDS_Long DDS_PublicationMatchedStatus::total_count_change

The changes in total_count since the last time the listener was called or the status was read.

Note that this number will never be negative (because it's the total number of times this DataWriter ever matched with a DataReader).

9.143.2.3 current_count

DDS_Long DDS_PublicationMatchedStatus::current_count

The current number of DataReaders with which this **DDSDataWriter** (p. 1589) is matched.

This number increases when a new match is discovered and decreases when an existing match goes away for any of the reasons described in **DDS_PublicationMatchedStatus** (p. 1273).

9.143.2.4 current_count_peak

DDS_Long DDS_PublicationMatchedStatus::current_count_peak

<<***extension***>> *(p. 432)* Greatest number of DataReaders that matched this **DDSDataWriter** (p. 1589) simultaneously.

That is, there was no moment in time when more than this many DataReaders matched this DataWriter. (As a result, total_count can be higher than current_count_peak.)

9.143.2.5 current_count_change

DDS_Long DDS_PublicationMatchedStatus::current_count_change

The change in current_count since the last time the listener was called or the status was read.

Note that a negative current_count_change means that one or more DataReaders have become unmatched for one or more of the reasons described in **DDS_PublicationMatchedStatus** (p. 1273).

9.143.2.6 last_subscription_handle

```
DDS_InstanceHandle_t DDS_PublicationMatchedStatus::last_subscription_handle
```

This InstanceHandle can be used to look up which remote **DDSDataReader** (p. 1556) was the last to cause this DataWriter's status to change, using **DDSDataWriter::get_matched_subscription_data** (p. 1600).

If the DataReader no longer matches this DataWriter due to any of the reasons in **DDS_PublicationMatchedStatus** (p. 1273) except incompatible QoS, then the DataReader has been purged from this DataWriter's DomainParticipant discovery database. (See the "Discovery Overview" section of the User's Manual.) In that case, the **DDSDataWriter::get_matched_subscription_data** (p. 1600) method will not be able to return information about the DataReader. The only way to get information about the lost DataReader is if you cached the information previously.

9.144 DDS_PublisherQos Struct Reference

QoS policies supported by a **DDSPublisher** (p. 1818) entity.

Public Member Functions

- bool **operator==** (const **DDS_PublisherQos** &r) const

  Compares two PublisherQos objects for equality.
- bool **operator!=** (const **DDS_PublisherQos** &r) const

  Compares two PublisherQos objects for inequality.
- **DDS_ReturnCode_t print** () const

  *Prints this **DDS_PublisherQos** (p. 1276) to stdout.*
- **DDS_ReturnCode_t to_string** (char *string, **DDS_UnsignedLong** &string_size) const

  *Obtains a string representation of this **DDS_PublisherQos** (p. 1276).*
- **DDS_ReturnCode_t to_string** (char *string, **DDS_UnsignedLong** &string_size, const **DDS_PublisherQos** &base) const

  *Obtains a string representation of this **DDS_PublisherQos** (p. 1276).*
- **DDS_ReturnCode_t to_string** (char *string, **DDS_UnsignedLong** &string_size, const **DDS_QosPrintFormat** &format) const

  *Obtains a string representation of this **DDS_PublisherQos** (p. 1276).*
- **DDS_ReturnCode_t to_string** (char *string, **DDS_UnsignedLong** &string_size, const **DDS_PublisherQos** &base, const **DDS_QosPrintFormat** &format) const

  *Obtains a string representation of this **DDS_PublisherQos** (p. 1276).*
- **DDS_ReturnCode_t to_string** (char *string, **DDS_UnsignedLong** &string_size, const **DDS_QosPrintAll_t** &) const

  *Obtains a string representation of this **DDS_PublisherQos** (p. 1276).*
- **DDS_ReturnCode_t to_string** (char *string, **DDS_UnsignedLong** &string_size, const **DDS_QosPrintAll_t** &, const **DDS_QosPrintFormat** &format) const

  *Obtains a string representation of this **DDS_PublisherQos** (p. 1276).*

Public Attributes

- struct **DDS_PresentationQosPolicy presentation**

  Presentation policy, ***PRESENTATION*** *(p. 615).*
- struct **DDS_PartitionQosPolicy partition**

  *<<**basic**>> (p. 201) Partition policy,* ***PARTITION*** *(p. 614).*
- struct **DDS_GroupDataQosPolicy group_data**

  Group data policy, ***GROUP_DATA*** *(p. 604).*
- struct **DDS_EntityFactoryQosPolicy entity_factory**

  *<<**basic**>> (p. 201) Entity factory policy,* ***ENTITY_FACTORY*** *(p. 600).*
- struct **DDS_AsynchronousPublisherQosPolicy asynchronous_publisher**

  *<<**extension**>> (p. 432) Asynchronous publishing settings for the* ***DDSPublisher*** *(p. 1818) and all entities that are created by it.*
- struct **DDS_EntityNameQosPolicy publisher_name**

  *<<**extension**>> (p. 432) EntityName policy,* ***ENTITY_NAME*** *(p. 601).*

9.144.1 Detailed Description

QoS policies supported by a **DDSPublisher** (p. 1818) entity.

You must set certain members in a consistent manner:

length of **DDS_GroupDataQosPolicy::value** (p. 1161) <= **DDS_DomainParticipantResourceLimitsQosPolicy**↩ **::publisher_group_data_max_length** (p. 992)

length of **DDS_PartitionQosPolicy::name** (p. 1242) <= **DDS_DomainParticipantResourceLimitsQosPolicy::max**↩ **_partitions** (p. 993)

combined number of characters (including terminating 0) in **DDS_PartitionQosPolicy::name** (p. 1242) <= **DDS_**↩ **DomainParticipantResourceLimitsQosPolicy::max_partition_cumulative_characters** (p. 993)

If any of the above are not true, **DDSPublisher::set_qos** (p. 1836) and **DDSPublisher::set_qos_with_profile** (p. 1837) will fail with **DDS_RETCODE_INCONSISTENT_POLICY** (p. 533) and **DDSDomainParticipant::create_**↩ **publisher** (p. 1644) will\ return NULL.

9.144.2 Member Function Documentation

9.144.2.1 operator==()

```
bool DDS_PublisherQos::operator== (
            const  DDS_PublisherQos & r ) const  [inline]
```

Compares two PublisherQos objects for equality.

See also

DDS_PublisherQos_equals (p. 108)

References **DDS_PublisherQos_equals()**.

9.144.2.2 operator"!=()

```
bool DDS_PublisherQos::operator!= (
            const  DDS_PublisherQos & r ) const  [inline]
```

Compares two PublisherQos objects for inequality.

See also

DDS_PublisherQos_equals (p. 108)

References **DDS_PublisherQos_equals()**.

9.144.3 Member Data Documentation

9.144.3.1 presentation

```
struct  DDS_PresentationQosPolicy DDS_PublisherQos::presentation
```

Presentation policy, **PRESENTATION** (p. 615).

9.144.3.2 partition

```
struct  DDS_PartitionQosPolicy DDS_PublisherQos::partition
```

<<***basic***>> *(p. 201)* Partition policy, **PARTITION** (p. 614).

9.144.3.3 group_data

```
struct  DDS_GroupDataQosPolicy DDS_PublisherQos::group_data
```

Group data policy, **GROUP_DATA** (p. 604).

9.144.3.4 entity_factory

`struct` **DDS_EntityFactoryQosPolicy** `DDS_PublisherQos::entity_factory`

<<***basic***>> *(p. 201)* Entity factory policy, **ENTITY_FACTORY** (p. 600).

9.144.3.5 asynchronous_publisher

`struct` **DDS_AsynchronousPublisherQosPolicy** `DDS_PublisherQos::asynchronous_publisher`

<<***extension***>> *(p. 432)* Asynchronous publishing settings for the **DDSPublisher** (p. 1818) and all entities that are created by it.

9.144.3.6 publisher_name

`struct` **DDS_EntityNameQosPolicy** `DDS_PublisherQos::publisher_name`

<<***extension***>> *(p. 432)* EntityName policy, **ENTITY_NAME** (p. 601).

9.145 DDS_PublishModeQosPolicy Struct Reference

<<***extension***>> *(p. 213)* Specifies how RTI Connext sends application data on the network. This QoS policy can be used to tell RTI Connext to use its *own* thread to send data, instead of the user thread.

Public Attributes

- **DDS_PublishModeQosPolicyKind kind**

  <<***basic***>> *(p. 201)* <<***extension***>> *(p. 213) Publishing mode*
- char * **flow_controller_name**

  <<***extension***>> *(p. 213) Name of the associated flow controller*
- **DDS_Long priority**

  <<***extension***>> *(p. 213) Publication priority*

9.145.1 Detailed Description

<<***extension***>> *(p. 213)* Specifies how RTI Connext sends application data on the network. This QoS policy can be used to tell RTI Connext to use its *own* thread to send data, instead of the user thread.

The publishing mode of a **DDSDataWriter** (p. 1589) determines whether data is written synchronously in the context of the user thread when calling **FooDataWriter::write** (p. 1951) or asynchronously in the context of a separate thread internal to the middleware.

Each **DDSPublisher** (p. 1818) spawns a single asynchronous publishing thread (**DDS_AsynchronousPublisherQos↩ Policy::thread** (p. 800)) to serve all its asynchronous **DDSDataWriter** (p. 1589) instances.

See also

DDS_AsynchronousPublisherQosPolicy (p. 798)

DDS_HistoryQosPolicy (p. 1162)

DDSFlowController (p. 1736)

Entity:

DDSDataWriter (p. 1589)

Properties:

RxO (p. 556) = N/A
Changeable (p. 556) = **NO** (p. 556)

9.145.2 Usage

The fastest way for RTI Connext to send data is for the user thread to execute the middleware code that actually sends the data itself. However, there are times when user applications may need or want an internal middleware thread to send the data instead.

When data is written asynchronously, a **DDSFlowController** (p. 1736), identified by `flow_controller_name`, can be used to shape the network traffic. Shaping a data flow usually means limiting the maximum data rates at which the middleware will send data for a **DDSDataWriter** (p. 1589). The flow controller will buffer any excess data and only send it when the send rate drops below the maximum rate. The flow controller's properties determine when the asynchronous publishing thread is allowed to send data and how much.

Asynchronous publishing may increase latency, but offers the following advantages:

- The **FooDataWriter::write** (p. 1951) call does not make any network calls and is therefore faster and more deterministic. This becomes important when the user thread is executing time-critical code.

- When data is written in bursts or when sending large data types as multiple fragments, a flow controller can throttle the send rate of the asynchronous publishing thread to avoid flooding the network.

- Asynchronously written samples for the same destination will be coalesced into a single network packet which reduces bandwidth consumption.

The maximum number of samples that will be coalesced depends on **NDDS_Transport_Property_t::gather_send_buffer_count_max** (p. 2047) (each sample requires at least 2-4 gather-send buffers). Performance can be improved by increasing **NDDS_Transport_Property_t::gather_send_buffer_count_max** (p. 2047). Note that the maximum value is operating system dependent.

The middleware must queue samples until they can be sent by the asynchronous publishing thread (as determined by the corresponding **DDSFlowController** (p. 1736)). The number of samples that will be queued is determined by the **DDS_HistoryQosPolicy** (p. 1162). When using **DDS_KEEP_LAST_HISTORY_QOS** (p. 604), only the most recent **DDS_HistoryQosPolicy::depth** (p. 1164) samples are kept in the queue. Once unsent samples are removed from the queue, they are no longer available to the asynchronous publishing thread and will therefore never be sent.

9.145.3 Member Data Documentation

9.145.3.1 kind

```
DDS_PublishModeQosPolicyKind DDS_PublishModeQosPolicy::kind
```

<<***basic***>> *(p. 201)* <<***extension***>> *(p. 213)* Publishing mode

[default] DDS_SYNCHRONOUS_PUBLISH_MODE_QOS (p. 629)

Categories

Immutable (p. 333), **Flow Control** (p. 282)

9.145.3.2 flow_controller_name

```
char* DDS_PublishModeQosPolicy::flow_controller_name
```

<<***extension***>> *(p. 213)* Name of the associated flow controller

The following builitin values are supported:

- **DDS_DEFAULT_FLOW_CONTROLLER_NAME** (p. 125)
- **DDS_FIXED_RATE_FLOW_CONTROLLER_NAME** (p. 126)
- **DDS_ON_DEMAND_FLOW_CONTROLLER_NAME** (p. 127)

NULL value or zero-length string refers to **DDS_DEFAULT_FLOW_CONTROLLER_NAME** (p. 125).

Warning

The value for this field can be one of the constants above or a string allocated with **DDS_String_dup()** (p. 761) to specify your own custom flow controller name. You should not assign a string literal. When you assign a new value with **DDS_String_dup()** (p. 761), first check if the current value is one of the constants above. If it is, simply replace it; if it's not, you have to release the current string and assign the new one. Here is an example of such an approach:

```
struct DDS_DataWriterQos writer_qos;

char *flow_controller_name = NULL;

DDS_DataWriterQos_initialize(&writer_qos);

...

DDS_Publisher_get_default_datawriter_qos(publisher, &writer_qos);

...

flow_controller_name = writer_qos.publish_mode.flow_controller_name;

if (flow_controller_name == DDS_DEFAULT_FLOW_CONTROLLER_NAME

        || flow_controller_name == DDS_FIXED_RATE_FLOW_CONTROLLER_NAME

        || flow_controller_name == DDS_ON_DEMAND_FLOW_CONTROLLER_NAME) {

    // Since the value was never on the heap, this won't leak memory

    flow_controller_name = NULL;

} else {

    // This means the value is present on the heap

    DDS_String_free(flow_controller_name);

}

// New value using DDS_String_dup() or one of the predefined constants

flow_controller_name = DDS_String_dup("My custom flow controller name");

...

DDS_DataWriterQos_finalize(&writer_qos);
```

The strings allocated with **DDS_String_dup()** (p. 761) are released when `writer_qos` is finalized.

More information about string handling can be found under **String Conventions** (p. 758).

See also

DDSDomainParticipant::create_flowcontroller (p. 1659)

[default] DDS_DEFAULT_FLOW_CONTROLLER_NAME (p. 125)

Categories

Immutable (p. 333), **Flow Control** (p. 282)

9.145.3.3 priority

```
DDS_Long DDS_PublishModeQosPolicy::priority
```

<<***extension***>> *(p. 213)* Publication priority

A positive integer value designating the relative priority of the **DDSDataWriter** (p. 1589), used to determine the transmission order of pending writes.

Use of publication priorities requires the asynchronous publisher (**DDS_ASYNCHRONOUS_PUBLISH_MODE_QOS** (p. 629)) with **DDS_FlowControllerProperty_t::scheduling_policy** (p. 1157) set to **DDS_HPF_FLOW_CONTROLLER_SCHED_POLICY** (p. 125).

Larger numbers have higher priority.

For multi-channel DataWriters, if the publication priority of any channel is set to any value other than **DDS_PUBLICATION_PRIORITY_UNDEFINED** (p. 628), then the channel's priority will take precedence over that of the DataWriter.

For multi-channel DataWriters, if the publication priority of any channel is **DDS_PUBLICATION_PRIORITY_UNDEFINED** (p. 628), then the channel will inherit the publication priority of the DataWriter.

If the publicaton priority of the DataWriter, and of any channel of a multi-channel DataWriter, are **DDS_PUBLICATION_PRIORITY_UNDEFINED** (p. 628), then the priority of the DataWriter or DataWriter channel will be assigned the lowest priority value.

If the publication priority of the DataWriter is **DDS_PUBLICATION_PRIORITY_AUTOMATIC** (p. 628), then the DataWriter will be assigned the priority of the largest publication priority of all samples in the DataWriter.

The publication priority of each sample can be set in the **DDS_WriteParams_t** (p. 1519) of the **FooDataWriter::write_w_params** (p. 1956) function.

For dispose and unregister samples, use the **DDS_WriteParams_t** (p. 1519) of **FooDataWriter::dispose_w_params** (p. 1959) and **FooDataWriter::unregister_instance_w_params** (p. 1951).

[default] DDS_PUBLICATION_PRIORITY_UNDEFINED (p. 628)

[range] [-1, MAX_INT]

Categories

Immutable (p. 333), **Flow Control** (p. 282)

9.146 DDS_QosPolicyCount Struct Reference

Type to hold a counter for a **DDS_QosPolicyId_t** (p. 557).

Public Attributes

- **DDS_QosPolicyId_t policy_id**

  The `QosPolicy` *ID.*
- **DDS_Long count**

  a counter

9.146.1 Detailed Description

Type to hold a counter for a **DDS_QosPolicyId_t** (p. 557).

9.146.2 Member Data Documentation

9.146.2.1 policy_id

DDS_QosPolicyId_t DDS_QosPolicyCount::policy_id

The `QosPolicy` ID.

9.146.2.2 count

DDS_Long DDS_QosPolicyCount::count

a counter

9.147 DDS_QosPolicyCountSeq Struct Reference

Declares IDL `sequence` < **DDS_QosPolicyCount** (p. 1283) >

9.147.1 Detailed Description

Declares IDL `sequence` < **DDS_QosPolicyCount** (p. 1283) >

Instantiates:

<<***generic***>> *(*p. *433)* **FooSeq** (p. 1965)

See also

DDS_QosPolicyCount (p. 1283)

9.148 DDS_QosPrintAll_t Struct Reference

Special type which is used to select the to_string overloads when printing Qos objects.

9.148.1 Detailed Description

Special type which is used to select the to_string overloads when printing Qos objects.

This sentinel value is used to select the to_string overload which prints the entirety of a Qos object (e.g., **DDS_DataReaderQos::to_string(char*, DDS_UnsignedLong&, const DDS_QosPrintAll_t&) const** (p. 150)). The only valid value for this type is the sentinel value **DDS_QOS_PRINT_ALL** (p. 169).

For example:

```
DataReaderQos qos;
char *str = NULL;
DDS_UnsignedLong str_size = 0;
qos.to_string(str, str_size, DDS_QOS_PRINT_ALL);
```

9.149 DDS_QosPrintFormat Struct Reference

A collection of attributes used to configure how a QoS appears when printed.

Public Attributes

- **DDS_Boolean is_standalone**

  Controls whether or not to print valid XML for this QoS policy (through the inclusion of a preamble).
- **DDS_Boolean print_private**

  Configures how private QoS policies should be printed.
- **DDS_UnsignedLong indent**

  Conifgures how much additional indent should be added to the string representation of a QoS.

9.149.1 Detailed Description

A collection of attributes used to configure how a QoS appears when printed.

To ensure that new objects of this type are initialized to a known value, assign them with the static initializer **DDS_QosPrintFormat_INITIALIZER** (p. 556).

9.149.2 Member Data Documentation

9.149.2.1 is_standalone

```
DDS_Boolean DDS_QosPrintFormat::is_standalone
```

Controls whether or not to print valid XML for this QoS policy (through the inclusion of a preamble).

The default value for this property (false) results in QoS being printed starting from the <ENTITY_qos> tag (where the <ENTITY_qos> tag is, e.g., <domain_participant_qos>, or, <datareader_qos>. This result cannot be directly parsed as valid XML (as it lacks <dds>, <qos_library>, and <qos_profile> tags). If is_standalone is set to true, these additional tags are printed, resulting in valid, parseable XML.

For example, with is_standalone set to false, a Qos may appear as:

```
<datareader_qos>

    <durability>

        <kind>TRANSIENT_LOCAL_DURABILITY_QOS</kind>

    </durability>

</datareader_qos>
```

Setting is_standalone to true would result in the same Qos being printed as:

```
<?xml version="1.0"?>

<dds xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">

    <qos_library name="QosLibrary">

        <qos_profile name="QosProfile">

            <datareader_qos>

                <durability>

                    <kind>TRANSIENT_LOCAL_DURABILITY_QOS</kind>

                </durability>

            </datareader_qos>

        </qos_profile>

    </qos_library>

</dds>
```

[default] false

9.149.2.2 print_private

```
DDS_Boolean DDS_QosPrintFormat::print_private
```

Configures how private QoS policies should be printed.

There are some QoS policies which are not intended for external use. By default, these QoS policies are only printed if they are different to the default value for that policy. When true, private policies are treated like any other policy, and printed if they are different to the supplied base QoS. These private policies cannot be parsed from XML, and are always printed as XML comments.

[default] false

9.149.2.3 indent

```
DDS_UnsignedLong DDS_QosPrintFormat::indent
```

Conifgures how much additional indent should be added to the string representation of a QoS.

Configures how much additional indent is applied when converting a QoS to a string. This value acts as a total offset on the string, increasing the indent which is applied to all elements by the same amount. With indent set to 0, a string representation of a QoS may appear as:

```
<datareader_qos>
    <durability>
        <kind>TRANSIENT_LOCAL_DURABILITY_QOS</kind>
    </durability>
</datareader_qos>
```

Setting the indent property to 1, the same QoS would be printed as:

```
    <datareader_qos>
        <durability>
            <kind>TRANSIENT_LOCAL_DURABILITY_QOS</kind>
        </durability>
    </datareader_qos>
```

I.e., the entire structure is indented.

[default] 0

9.150 DDS_QueryConditionParams Struct Reference

<<***extension***>> *(*p. *432)* Input parameters for **DDSDataReader::create_querycondition_w_params** (p. 1562)

Public Attributes

- struct **DDS_ReadConditionParams as_readconditionparams**
  Read condition parameters.
- char * **query_expression**
  Expression for the query.
- struct **DDS_StringSeq query_parameters**
  Parameters for the query expression.

9.150.1 Detailed Description

<<***extension***>> *(p. 432)* Input parameters for **DDSDataReader::create_querycondition_w_params** (p. 1562)

9.150.2 Member Data Documentation

9.150.2.1 as_readconditionparams

`struct` **`DDS_ReadConditionParams`** `DDS_QueryConditionParams::as_readconditionparams`

Read condition parameters.

9.150.2.2 query_expression

`char* DDS_QueryConditionParams::query_expression`

Expression for the query.

9.150.2.3 query_parameters

`struct` **`DDS_StringSeq`** `DDS_QueryConditionParams::query_parameters`

Parameters for the query expression.

9.151 DDS_ReadConditionParams Struct Reference

<<***extension***>> *(p. 432)* Input parameters for **DDSDataReader::create_readcondition_w_params** (p. 1561)

Public Attributes

- **DDS_SampleStateMask sample_states**
  Sample state.
- **DDS_ViewStateMask view_states**
  View state.
- **DDS_InstanceStateMask instance_states**
  Instance state.
- **DDS_StreamKindMask stream_kinds**
  Stream kind.

9.151.1 Detailed Description

<<***extension***>> *(p. 432)* Input parameters for **DDSDataReader::create_readcondition_w_params** (p. 1561)

9.151.2 Member Data Documentation

9.151.2.1 sample_states

DDS_SampleStateMask DDS_ReadConditionParams::sample_states

Sample state.

Sample state of the data samples that are of interest.

9.151.2.2 view_states

DDS_ViewStateMask DDS_ReadConditionParams::view_states

View state.

View state of the data samples that are of interest.

9.151.2.3 instance_states

DDS_InstanceStateMask DDS_ReadConditionParams::instance_states

Instance state.

Instance state of the data samples that are of interest.

9.151.2.4 stream_kinds

DDS_StreamKindMask DDS_ReadConditionParams::stream_kinds

Stream kind.

Stream kind of the data samples that are of interest.

9.152 DDS_ReaderDataLifecycleQosPolicy Struct Reference

<<***spec***>> *(p. 209)* Controls how a DataReader manages the lifecycle of the data that it has received.

Public Attributes

- struct **DDS_Duration_t autopurge_nowriter_samples_delay**

  *<<**spec**>> (p. 209) Minimum duration for which the **DDSDataReader** (p. 1556) will maintain samples for an instance once its* `instance_state` *becomes **DDS_NOT_ALIVE_NO_WRITERS_INSTANCE_STATE** (p. 166).*
- struct **DDS_Duration_t autopurge_disposed_samples_delay**

  *<<**spec**>> (p. 209) Minimum duration for which the **DDSDataReader** (p. 1556) will maintain samples for an instance once its* `instance_state` *becomes **DDS_NOT_ALIVE_DISPOSED_INSTANCE_STATE** (p. 166).*
- struct **DDS_Duration_t autopurge_disposed_instances_delay**

  *<<**extension**>> (p. 213) Minimum duration for which the **DDSDataReader** (p. 1556) will maintain information about a received instance once its* `instance_state` *becomes **DDS_NOT_ALIVE_DISPOSED_INSTANCE_STATE** (p. 166) and there are no samples for the instance in the DataReader queue.*
- struct **DDS_Duration_t autopurge_nowriter_instances_delay**

  *<<**extension**>> (p. 213) Minimum duration for which the **DDSDataReader** (p. 1556) will maintain information about a received instance once its* `instance_state` *becomes **DDS_NOT_ALIVE_NO_WRITERS_INSTANCE_STATE** (p. 166) and there are no samples for the instance in the DataReader queue.*

9.152.1 Detailed Description

*<<**spec**>> (p. 209)* Controls how a DataReader manages the lifecycle of the data that it has received.

When a DataReader receives data, it is stored in a receive queue for the DataReader. The user application may either take the data from the queue or leave it there.

This QoS policy controls whether or not RTI Connext will automatically remove data from the receive queue (so that user applications cannot access it afterwards) when it detects that there are no more DataWriters alive for that data. It specifies how long a **DDSDataReader** (p. 1556) must retain information regarding instances that have the instance_state **DDS_NOT_ALIVE_NO_WRITERS_INSTANCE_STATE** (p. 166).

Note: This policy is not concerned with keeping reliable reader state or discovery information.

The **DDSDataReader** (p. 1556) internally maintains the samples that have not been "taken" by the application, subject to the constraints imposed by other QoS policies such as **DDS_HistoryQosPolicy** (p. 1162) and **DDS_ResourceLimits↩ QosPolicy** (p. 1308).

The **DDSDataReader** (p. 1556) also maintains information regarding the identity, `view_state` and `instance_↩ state` of data instances even after all samples have been taken. This is needed to properly compute the states when future samples arrive.

Under normal circumstances the **DDSDataReader** (p. 1556) can only reclaim all resources for instances for which there are no writers and for which all samples have been 'taken'. The last sample the **DDSDataReader** (p. 1556) will have taken for that instance will have an `instance_state` of either **DDS_NOT_ALIVE_NO_WRITERS_INSTANCE_↩ STATE** (p. 166) or **DDS_NOT_ALIVE_DISPOSED_INSTANCE_STATE** (p. 166) depending on whether or not the last writer that had ownership of the instance disposed it.

In the absence of **READER_DATA_LIFECYCLE** (p. 630), this behavior could cause problems if the application forgets to take those samples. "Untaken" samples will prevent the **DDSDataReader** (p. 1556) from reclaiming the resources and they would remain in the **DDSDataReader** (p. 1556) indefinitely.

A DataReader can also reclaim all resources for instances that have an instance state of **DDS_NOT_ALIVE_↩ DISPOSED_INSTANCE_STATE** (p. 166) and for which all DDS samples have been 'taken'. DataReaders will only reclaim resources in this situation when the autopurge_disposed_instances_delay has been set to zero.

For keyed Topics, the consideration of removing data samples from the receive queue is done on a per instance (key) basis. Thus when RTI Connext detects that there are no longer DataWriters alive for a certain key value of a Topic (an instance of the Topic), it can be configured to remove all data samples for that instance (key).

Entity:

DDSDataReader (p. 1556)

Properties:

RxO (p. 556) = N/A
Changeable (p. 556) = **YES** (p. 556)

9.152.2 Member Data Documentation

9.152.2.1 autopurge_nowriter_samples_delay

`struct` **`DDS_Duration_t`** `DDS_ReaderDataLifecycleQosPolicy::autopurge_nowriter_samples_delay`

<<***spec***>> *(p. 209)* Minimum duration for which the **DDSDataReader** (p. 1556) will maintain samples for an instance once its `instance_state` becomes **DDS_NOT_ALIVE_NO_WRITERS_INSTANCE_STATE** (p. 166).

At some point after this time elapses, the **DDSDataReader** (p. 1556) will purge all samples for the instance; any "untaken" samples will also be dropped.

[default] DDS_DURATION_INFINITE (p. 522)

[range] [1 nanosec, 1 year] or **DDS_DURATION_INFINITE** (p. 522)

Categories

Immutable (p. 333), **Instance Lifecycle** (p. 286), **Sample Memory Management** (p. 321), **Data Availability** (p. 261)

9.152.2.2 autopurge_disposed_samples_delay

`struct` **`DDS_Duration_t`** `DDS_ReaderDataLifecycleQosPolicy::autopurge_disposed_samples_delay`

<<***spec***>> *(p. 209)* Minimum duration for which the **DDSDataReader** (p. 1556) will maintain samples for an instance once its `instance_state` becomes **DDS_NOT_ALIVE_DISPOSED_INSTANCE_STATE** (p. 166).

After this time elapses, the **DDSDataReader** (p. 1556) will purge all samples for the instance even if they have not been read by the application. This purge is done lazily when space is needed for other samples or instances.

[default] DDS_DURATION_INFINITE (p. 522)

[range] [1 nanosec, 1 year] or **DDS_DURATION_INFINITE** (p. 522)

Categories

Immutable (p. 333), **Instance Lifecycle** (p. 286), **Sample Memory Management** (p. 321), **Data Availability** (p. 261)

9.152.2.3 autopurge_disposed_instances_delay

`struct` **`DDS_Duration_t`** `DDS_ReaderDataLifecycleQosPolicy::autopurge_disposed_instances_delay`

<<***extension***>> *(p. 213)* Minimum duration for which the **DDSDataReader** (p. 1556) will maintain information about a received instance once its `instance_state` becomes **DDS_NOT_ALIVE_DISPOSED_INSTANCE_STATE** (p. 166) and there are no samples for the instance in the DataReader queue.

After this time elapses, when the last sample for the disposed instance is taken, the **DDSDataReader** (p. 1556) will keep only the minimum state about the instance.

If you do not want to keep this minimum state after the delay period, also set keep_minimum_state_for_instances to FALSE in **DDS_DataReaderResourceLimitsQosPolicy** (p. 867).

The only currently supported values are **DDS_DURATION_ZERO** (p. 523) and **DDS_DURATION_INFINITE** (p. 522). A value of **DDS_DURATION_ZERO** (p. 523) will purge an instance's state immediately after the instance state transitions to **DDS_NOT_ALIVE_DISPOSED_INSTANCE_STATE** (p. 166), as long as all samples, including the dispose sample, associated with that instance have been 'taken'.

[default] DDS_DURATION_INFINITE (p. 522)

Categories

Immutable (p. 333), **Instance Lifecycle** (p. 286), **Sample Memory Management** (p. 321), **Data Availability** (p. 261)

9.152.2.4 autopurge_nowriter_instances_delay

`struct` **`DDS_Duration_t`** `DDS_ReaderDataLifecycleQosPolicy::autopurge_nowriter_instances_delay`

<<***extension***>> *(p. 213)* Minimum duration for which the **DDSDataReader** (p. 1556) will maintain information about a received instance once its `instance_state` becomes **DDS_NOT_ALIVE_NO_WRITERS_INSTANCE_STATE** (p. 166) and there are no samples for the instance in the DataReader queue.

An instance will transition to the **DDS_NOT_ALIVE_NO_WRITERS_INSTANCE_STATE** (p. 166) `instance_state` when all known writers for the instance have either lost liveliness or have unregistered themselves from the instance. After this time elapses, when the last sample for the instance without writers is taken, the **DDSDataReader** (p. 1556) will keep only the minimum state about the instance.

If you do not want to keep this minimum state after the delay period, also set keep_minimum_state_for_instances to FALSE in **DDS_DataReaderResourceLimitsQosPolicy** (p. 867).

The only currently supported values are **DDS_DURATION_ZERO** (p. 523) and **DDS_DURATION_INFINITE** (p. 522). A value of **DDS_DURATION_ZERO** (p. 523) will purge an instance's state immediately after the instance state transitions to **DDS_NOT_ALIVE_NO_WRITERS_INSTANCE_STATE** (p. 166), as long as all samples, including the no_writers sample, associated with that instance have been 'taken'.

[default] DDS_DURATION_ZERO (p. 523)

Categories

Immutable (p. 333), **Instance Lifecycle** (p. 286), **Sample Memory Management** (p. 321), **Data Availability** (p. 261)

9.153 DDS_ReceiverPoolQosPolicy Struct Reference

*<<**extension**>> (p. 213)* Configures threads used by RTI Connext to receive and process data from transports (for example, UDP sockets).

Public Attributes

- struct **DDS_ThreadSettings_t thread**

  *<<**extension**>> (p. 213) Receiver pool thread(s).*
- **DDS_Long buffer_size**

  *<<**extension**>> (p. 213) The receive buffer size in bytes.*
- **DDS_Long buffer_alignment**

  [DEPRECATED] *(p. **384**) <<**extension**>> (p. 213) The receive buffer alignment.*

9.153.1 Detailed Description

*<<**extension**>> (p. 213)* Configures threads used by RTI Connext to receive and process data from transports (for example, UDP sockets).

This QoS policy is an extension to the DDS standard.

Entity:

DDSDomainParticipant (p. 1619)

Properties:

RxO (p. 556) = N/A
Changeable (p. 556) = **NO** (p. 556)

See also

Controlling CPU Core Affinity for RTI Threads (p. 548)

9.153.2 Usage

This QoS policy sets the thread properties such as priority level and stack size for the threads used by the middleware to receive and process data from transports.

RTI uses a separate receive thread per port per transport plug-in. To force RTI Connext to use a separate thread to process the data for a **DDSDataReader** (p. 1556), set a unique port for the **DDS_TransportUnicastQosPolicy** (p. 1422) or **DDS_TransportMulticastQosPolicy** (p. 1415) for the **DDSDataReader** (p. 1556).

This QoS policy also sets the size of the buffer used to store packets received from a transport. This buffer size will limit the largest single packet of data that a **DDSDomainParticipant** (p. 1619) will accept from a transport. Users will often set this size to the largest packet that any of the transports used by their application will deliver. For many applications, the value 65,536 (64 K) is a good choice; this value is the largest packet that can be sent/received via UDP.

9.153.3 Member Data Documentation

9.153.3.1 thread

`struct` **DDS_ThreadSettings_t** DDS_ReceiverPoolQosPolicy::thread

<<***extension***>> *(p. 213)* Receiver pool thread(s).

There is at least one receive thread, possibly more.

[default] priority above normal.
The actual value depends on your architecture:

For Windows: 2
For Linux: OS default priority
For a complete list of platform specific values, please refer to `Platform Notes`.

[default] The actual value depends on your architecture:

For Windows: OS default stack size
For Linux: OS default stack size
For a complete list of platform specific values, please refer to `Platform Notes`.

[default] mask **DDS_THREAD_SETTINGS_FLOATING_POINT** (p. 548) | **DDS_THREAD_SETTINGS_STDIO** (p. 548)

Categories

Immutable (p. 333), **Concurrency** (p. 258)

9.153.3.2 buffer_size

DDS_Long DDS_ReceiverPoolQosPolicy::buffer_size

<<***extension***>> *(p. 213)* The receive buffer size in bytes.

The receive buffer is used by the receive thread to store the raw data that arrives over the transports in non-zero-copy transports.

Zero-copy transports do not copy their data into the buffer provided by the receive thread. Instead, they provide the receive thread data in buffers allocated by the transports themselves. Only the shared memory built-in transport (SHMEM) supports zero-copy.

buffer_size must always be at least as large as the maximum **NDDS_Transport_Property_t::message_size_max** (p. 2047) across *all* of the transports being used that are not doing zero-copy.

By default (**DDS_LENGTH_AUTO** (p. 631)): the size is equal to the maximum **NDDS_Transport_Property_t↩ ::message_size_max** (p. 2047) across *all* of the non-zero-copy transports.

You may want the value to be greater than the default if you try to limit the largest data packet that can be sent through the transport(s) in one application, but you still want to receive data from other applications that have not made the same change.

For example, to avoid IP fragmentation, you may want to set **NDDS_Transport_Property_t::message_size_max** (p. 2047) for IP-based transports to a small value, such as 1400 bytes. However, you may not be able to apply this change to all the applications at the same time. To receive data from these other applications the buffer_size should be equal to the original **NDDS_Transport_Property_t::message_size_max** (p. 2047).

[default] DDS_LENGTH_AUTO (p. 631)

[range] [1, 1 GB] or **DDS_LENGTH_AUTO** (p. 631)

Categories

Immutable (p. 333), **Middleware Memory Management** (p. 297)

9.153.3.3 buffer_alignment

```
DDS_Long DDS_ReceiverPoolQosPolicy::buffer_alignment
```

[DEPRECATED] (p. **384)** <<***extension***>> *(*p. *213)* The receive buffer alignment.

Most users will not need to change this alignment.

[default] 16

[range] [1,1024] Value must be a power of 2.

Categories

Immutable (p. 333), **Middleware Memory Management** (p. 297)

9.154 DDS_ReliabilityQosPolicy Struct Reference

<<***extension***>> *(*p. *213)* <<***spec***>> *(*p. *209)* Indicates the level of reliability offered/requested by RTI Connext.

Public Attributes

- **DDS_ReliabilityQosPolicyKind kind**

  <<***basic***>> *(p. 201)* <<***spec***>> *(p. 209) Kind of reliability.*
- struct **DDS_Duration_t max_blocking_time**

  <<***basic***>> *(p. 201)* <<***spec***>> *(p. 209) The maximum time a DataWriter can block on a write() call.*
- **DDS_ReliabilityQosPolicyAcknowledgmentModeKind acknowledgment_kind**

  <<***extension***>> *(p. 213) Kind of reliable acknowledgment*
- **DDS_InstanceStateConsistencyKind instance_state_consistency_kind**

  <<***extension***>> *(p. 213) Whether instance state consistency is enabled*
- **DDS_Boolean exact_reliability_kind_match**

  <<***extension***>> *(p. 213) Whether an exact reliability kind match is required for endpoint matching.*

9.154.1 Detailed Description

<<***extension***>> *(p. 213)* <<***spec***>> *(p. 209)* Indicates the level of reliability offered/requested by RTI Connext.

Entity:

DDSTopic (p. 1885), **DDSDataReader** (p. 1556), **DDSDataWriter** (p. 1589)

Status:

DDS_OFFERED_INCOMPATIBLE_QOS_STATUS (p. 540), **DDS_REQUESTED_INCOMPATIBLE_QOS_↩ STATUS** (p. 540)

Properties:

RxO (p. 556) = YES
Changeable (p. 556) = **UNTIL ENABLE** (p. 556) (the instance_state_consistency_kind is **Changeable** (p. 556) = **NO** (p. 556))

9.154.2 Usage

This policy indicates the level of reliability requested by a **DDSDataReader** (p. 1556) or offered by a **DDSDataWriter** (p. 1589).

The reliability of a connection between a DataWriter and DataReader is entirely user configurable. It can be done on a per DataWriter/DataReader connection. A connection may be configured to be "best effort" which means that RTI Connext will not use any resources to monitor or guarantee that the data sent by a DataWriter is received by a DataReader.

For some use cases, such as the periodic update of sensor values to a GUI displaying the value to a person, **DDS↩ _BEST_EFFORT_RELIABILITY_QOS** (p. 633) delivery is often good enough. It is certainly the fastest, most efficient, and least resource-intensive (CPU and network bandwidth) method of getting the newest/latest value for a topic from

DataWriters to DataReaders. But there is no guarantee that the data sent will be received. It may be lost due to a variety of factors, including data loss by the physical transport such as wireless RF or even Ethernet.

However, there are data streams (topics) in which you want an absolute guarantee that all data sent by a DataWriter is received reliably by DataReaders. This means that RTI Connext must check whether or not data was received, and repair any data that was rejected by resending a copy of the data as many times as it takes for the DataReader to receive the data. RTI Connext uses a reliability protocol configured and tuned by these QoS policies: **DDS_HistoryQosPolicy** (p. 1162), **DDS_DataWriterProtocolQosPolicy** (p. 892), **DDS_DataReaderProtocolQosPolicy** (p. 843), and **DDS_↩ ResourceLimitsQosPolicy** (p. 1308).

The Reliability QoS policy is simply a switch to turn on the reliability protocol for a DataWriter/DataReader connection. The level of reliability provided by RTI Connext is determined by the configuration of the aforementioned QoS policies.

You can configure RTI Connext to deliver *all* data in the order they were sent (also known as absolute or strict reliability). Or, as a tradeoff for less memory, CPU, and network usage, you can choose a reduced level of reliability where only the last *N* values are guaranteed to be delivered reliably to DataReaders (where *N* is user-configurable). In the reduced level of reliability, there are no guarantees that the data sent before the last *N* are received. Only the last *N* data packets are monitored and repaired if necessary.

These levels are ordered, **DDS_BEST_EFFORT_RELIABILITY_QOS** (p. 633) < **DDS_RELIABLE_RELIABILITY_↩ QOS** (p. 633). A **DDSDataWriter** (p. 1589) offering one level is implicitly offering all levels below.

The setting of this policy has a dependency on the setting of the **RESOURCE_LIMITS** (p. 635) policy. In case the reliability kind is set to **DDS_RELIABLE_RELIABILITY_QOS** (p. 633) the write operation on the **DDSDataWriter** (p. 1589) may block if the modification would cause data to be lost or else cause one of the limits in specified in the **RESOURCE↩ _LIMITS** (p. 635) to be exceeded. Under these circumstances, the **RELIABILITY** (p. 631) `max_blocking_time` configures the maximum duration the write operation may block.

If the **DDS_ReliabilityQosPolicy::kind** (p. 1298) is set to **DDS_RELIABLE_RELIABILITY_QOS** (p. 633), data samples originating from a single **DDSDataWriter** (p. 1589) cannot be made available to the **DDSDataReader** (p. 1556) if there are previous data samples that have not been received yet due to a communication error. In other words, RTI Connext will repair the error and resend data samples as needed in order to reconstruct a correct snapshot of the **DDSDataWriter** (p. 1589) history before it is accessible by the **DDSDataReader** (p. 1556).

If the **DDS_ReliabilityQosPolicy::kind** (p. 1298) is set to **DDS_BEST_EFFORT_RELIABILITY_QOS** (p. 633), the service will not re-transmit missing data samples. However, for data samples originating from any one DataWriter the service will ensure they are stored in the **DDSDataReader** (p. 1556) history in the same order they originated in the **DDSDataWriter** (p. 1589). In other words, the **DDSDataReader** (p. 1556) may miss some data samples, but it will never see the value of a data object change from a newer value to an older value.

See also

DDS_HistoryQosPolicy (p. 1162)

DDS_ResourceLimitsQosPolicy (p. 1308)

9.154.3 Compatibility

The value offered is considered compatible with the value requested if and only if:

- The inequality *offered* `kind` >= *requested* `kind` evaluates to 'TRUE'. For the purposes of this inequality, the values of **DDS_ReliabilityQosPolicy::kind** (p. 1298) are considered ordered such that **DDS_BEST_EFFORT↩ _RELIABILITY_QOS** (p. 633) < **DDS_RELIABLE_RELIABILITY_QOS** (p. 633). If **DDS_ReliabilityQosPolicy↩ ::exact_reliability_kind_match** (p. 1299) is set to true, then only exact kind matches, *offered* `kind` == *requested* `kind`, are considered compatible.

- The offered acknowledgment_kind = **DDS_PROTOCOL_ACKNOWLEDGMENT_MODE** (p. 633) and requested acknowledgment_kind = **DDS_PROTOCOL_ACKNOWLEDGMENT_MODE** (p. 633) OR offered acknowledgment_kind != **DDS_PROTOCOL_ACKNOWLEDGMENT_MODE** (p. 633).

- The inequality *offered* `instance_state_consistency_kind` >= *requested* `instance_state_consistency_kind` evaluates to 'TRUE'. For the purposes of this inequaility, the values of **DDS_ReliabilityQosPolicy::instance_state_consistency_kind** (p. 1299) are considered ordered such that **DDS_NO_RECOVER_INSTANCE_STATE_CONSISTENCY** (p. 634) < **DDS_RECOVER_INSTANCE_STATE_CONSISTENCY** (p. 634).

9.154.4 Member Data Documentation

9.154.4.1 kind

DDS_ReliabilityQosPolicyKind DDS_ReliabilityQosPolicy::kind

<<***basic***>> *(p. 201)* <<***spec***>> *(p. 209)* Kind of reliability.

[default] DDS_BEST_EFFORT_RELIABILITY_QOS (p. 633) for **DDSDataReader** (p. 1556) and **DDSTopic** (p. 1885), **DDS_RELIABLE_RELIABILITY_QOS** (p. 633) for **DDSDataWriter** (p. 1589)

Categories

Immutable (p. 333), **Matching** (p. 382), **Discovery Metadata** (p. 377), **Reliability** (p. 314)

9.154.4.2 max_blocking_time

struct **DDS_Duration_t** DDS_ReliabilityQosPolicy::max_blocking_time

<<***basic***>> *(p. 201)* <<***spec***>> *(p. 209)* The maximum time a DataWriter can block on a write() call.

See `Writing Data`, in the Core Libraries User's Manual, for more information about when the write() call can block and what happens when it times out.

Has no meaning for DataReaders.

[default] 100 milliseconds

[range] [0,1 year] or **DDS_DURATION_INFINITE** (p. 522)

See also

DDS_ResourceLimitsQosPolicy (p. 1308)

Categories

Immutable (p. 333), **Discovery Metadata** (p. 377), **Reliability** (p. 314), **Flow Control** (p. 282)

9.154.4.3 acknowledgment_kind

DDS_ReliabilityQosPolicyAcknowledgmentModeKind DDS_ReliabilityQosPolicy::acknowledgment_kind

<<***extension***>> *(p. 213)* Kind of reliable acknowledgment

This setting applies only to the case where **DDS_ReliabilityQosPolicy::kind** (p. 1298) = **DDS_RELIABLE_↩ RELIABILITY_QOS** (p. 633).

Sets the kind acknowledgments supported by a **DDSDataWriter** (p. 1589) and sent by **DDSDataReader** (p. 1556).

[default] DDS_PROTOCOL_ACKNOWLEDGMENT_MODE (p. 633)

Categories

Immutable (p. 333), **Matching** (p. 382), **Discovery Metadata** (p. 377), **Reliability** (p. 314), **Application Acknowledgement** (p. 253)

9.154.4.4 instance_state_consistency_kind

DDS_InstanceStateConsistencyKind DDS_ReliabilityQosPolicy::instance_state_consistency_kind

<<***extension***>> *(p. 213)* Whether instance state consistency is enabled

A DataReader that determines that the DataWriter is no longer alive will transition instances to NOT_ALIVE_NO_↩ WRITERS if there are no other DataWriters updating that instance. If the DataReader rediscovers the DataWriter, the DataReader does not automatically transition instances back to the state they were in prior to the disconnection until it gets updates (i.e., samples) for them.

If `InstanceStateConsistencyKind` is set to `RECOVER_INSTANCE_STATE_CONSISTENCY`, then when the DataReader rediscovers a DataWriter, the DataReader will query the DataWriter for the state of its instances, and restore the instances to their correct state.

[default] DDS_NO_RECOVER_INSTANCE_STATE_CONSISTENCY (p. 634) for **DDSDataReader** (p. 1556), **DDS_↩ RECOVER_INSTANCE_STATE_CONSISTENCY** (p. 634) for **DDSDataWriter** (p. 1589)

Categories

Immutable (p. 333), **Matching** (p. 382), **Discovery Metadata** (p. 377), **Reliability** (p. 314), **Instance Lifecycle** (p. 286)

9.154.4.5 exact_reliability_kind_match

```
DDS_Boolean DDS_ReliabilityQosPolicy::exact_reliability_kind_match
```

<<***extension***>> *(p. 213)* Whether an exact reliability kind match is required for endpoint matching.

If set to TRUE in the DataWriter, the offered and requested reliability kinds must be exactly the same to be considered compatible for endpoint matching. If set to FALSE in the DataWriter, the offered reliability kind must be greater than or equal to the requested reliability kind to be considered compatible for endpoint matching. Set it to TRUE in the DataWriter to save memory and CPU cycles if communication with **DDS_BEST_EFFORT_RELIABILITY_QOS** (p. 633) DataReaders is not required.

This setting has no effect on DataReaders.

[default] FALSE

Categories

Immutable (p. 333), **Matching** (p. 382), **Discovery Metadata** (p. 377), **Reliability** (p. 314)

9.155 DDS_ReliableReaderActivityChangedStatus Struct Reference

<<***extension***>> *(p. 432)* Describes the activity (i.e. are acknowledgements forthcoming) of reliable readers matched to a reliable writer.

Public Attributes

- **DDS_Long active_count**

  The current number of reliable readers currently matched with this reliable writer.
- **DDS_Long inactive_count**

  The number of reliable readers that have been dropped by this reliable writer because they failed to send acknowledgements in a timely fashion.
- **DDS_Long active_count_change**

  The most recent change in the number of active remote reliable readers.
- **DDS_Long inactive_count_change**

  The most recent change in the number of inactive remote reliable readers.
- **DDS_InstanceHandle_t last_instance_handle**

  The instance handle of the last reliable remote reader to be determined inactive.

9.155.1 Detailed Description

<<***extension***>> *(*p. *432)* Describes the activity (i.e. are acknowledgements forthcoming) of reliable readers matched to a reliable writer.

Entity:

DDSDataWriter (p. 1589)

Listener:

DDSDataWriterListener (p. 1612)

This status is the reciprocal status to the **DDS_LivelinessChangedStatus** (p. 1176) on the reader. It is different than the **DDS_LivelinessLostStatus** (p. 1179) on the writer in that the latter informs the writer about its own liveliness; this status informs the writer about the "liveliness" (activity) of its matched readers.

All counts in this status will remain at zero for best effort writers.

9.155.2 Member Data Documentation

9.155.2.1 active_count

```
DDS_Long DDS_ReliableReaderActivityChangedStatus::active_count
```

The current number of reliable readers currently matched with this reliable writer.

9.155.2.2 inactive_count

```
DDS_Long DDS_ReliableReaderActivityChangedStatus::inactive_count
```

The number of reliable readers that have been dropped by this reliable writer because they failed to send acknowledgements in a timely fashion.

A reader is considered to be inactive after is has been sent heartbeats **DDS_RtpsReliableWriterProtocol_t::max_↩ heartbeat_retries** (p. 1326) times, each heartbeat having been separated from the previous by the current heartbeat period.

9.155.2.3 active_count_change

```
DDS_Long DDS_ReliableReaderActivityChangedStatus::active_count_change
```

The most recent change in the number of active remote reliable readers.

9.155.2.4 inactive_count_change

```
DDS_Long DDS_ReliableReaderActivityChangedStatus::inactive_count_change
```

The most recent change in the number of inactive remote reliable readers.

9.155.2.5 last_instance_handle

```
DDS_InstanceHandle_t DDS_ReliableReaderActivityChangedStatus::last_instance_handle
```

The instance handle of the last reliable remote reader to be determined inactive.

9.156 DDS_ReliableWriterCacheChangedStatus Struct Reference

<<***extension***>> *(*p. *432)* A summary of the state of a data writer's cache of unacknowledged samples written.

Public Attributes

- struct **DDS_ReliableWriterCacheEventCount empty_reliable_writer_cache**

  The number of times the reliable writer's cache of unacknowledged samples has become empty.
- struct **DDS_ReliableWriterCacheEventCount full_reliable_writer_cache**

  The number of times the reliable writer's cache, or send window, of unacknowledged samples has become full.
- struct **DDS_ReliableWriterCacheEventCount low_watermark_reliable_writer_cache**

  The number of times the reliable writer's cache of unacknowledged samples has fallen to the low watermark.
- struct **DDS_ReliableWriterCacheEventCount high_watermark_reliable_writer_cache**

  The number of times the reliable writer's cache of unacknowledged samples has risen to the high watermark.
- **DDS_Long unacknowledged_sample_count**

  The current number of unacknowledged samples in the writer's cache.
- **DDS_Long unacknowledged_sample_count_peak**

  The highest value that unacknowledged_sample_count has reached until now.
- **DDS_LongLong replaced_unacknowledged_sample_count**

  The number of unacknowledged samples that have been replaced in the writer's cache.

9.156.1 Detailed Description

<<***extension***>> *(*p. *432)* A summary of the state of a data writer's cache of unacknowledged samples written.

Entity:

DDSDataWriter (p. 1589)

Listener:

DDSDataWriterListener (p. 1612)

A written sample is unacknowledged (and therefore accounted for in this status) if the writer is reliable and one or more readers matched with the writer has not yet sent an acknowledgement to the writer declaring that it has received the sample.

If the low watermark is zero and the unacknowledged sample count decreases to zero, both the low watermark and cache empty events are considered to have taken place. A single callback will be dispatched (assuming the user has requested one) that contains both status changes. The same logic applies when the high watermark is set equal to the maximum number of samples and the cache becomes full.

9.156.2 Member Data Documentation

9.156.2.1 empty_reliable_writer_cache

`struct` **`DDS_ReliableWriterCacheEventCount`** `DDS_ReliableWriterCacheChangedStatus::empty_reliable_writer_cache`

The number of times the reliable writer's cache of unacknowledged samples has become empty.

9.156.2.2 full_reliable_writer_cache

`struct` **`DDS_ReliableWriterCacheEventCount`** `DDS_ReliableWriterCacheChangedStatus::full_reliable_writer_cache`

The number of times the reliable writer's cache, or send window, of unacknowledged samples has become full.

Applies to writer's cache when the send window is enabled (when both **DDS_RtpsReliableWriterProtocol_t::min_send_window_size** (p. 1333) and **DDS_RtpsReliableWriterProtocol_t::max_send_window_size** (p. 1333) are not **DDS_LENGTH_UNLIMITED** (p. 636)).

Otherwise, applies when the number of unacknowledged samples has reached the send window limit.

9.156.2.3 low_watermark_reliable_writer_cache

```
struct DDS_ReliableWriterCacheEventCount DDS_ReliableWriterCacheChangedStatus::low_watermark_↩
reliable_writer_cache
```

The number of times the reliable writer's cache of unacknowledged samples has fallen to the low watermark.

A low watermark event will only be considered to have taken place when the number of unacknowledged samples in the writer's cache *decreases* to this value. A sample count that increases to this value will not result in a callback or in a change to the total count of low watermark events.

When the writer's send window is enabled, the low watermark is scaled down, if necessary, to fit within the current send window.

Set the low watermark value using the **DDS_RtpsReliableWriterProtocol_t::low_watermark** (p. 1321) field of the **DDS_RtpsReliableWriterProtocol_t** (p. 1320) structure.

9.156.2.4 high_watermark_reliable_writer_cache

```
struct DDS_ReliableWriterCacheEventCount DDS_ReliableWriterCacheChangedStatus::high_watermark_↩
reliable_writer_cache
```

The number of times the reliable writer's cache of unacknowledged samples has risen to the high watermark.

A high watermark event will only be considered to have taken place when the number of unacknowledged sampled *increases* to this value. A sample count that was above this value and then decreases back to it will not trigger an event.

When the writer's send window is enabled, the high watermark is scaled down, if necessary, to fit within the current send window.

Set the high watermark value using the **DDS_RtpsReliableWriterProtocol_t::high_watermark** (p. 1322) field of the **DDS_RtpsReliableWriterProtocol_t** (p. 1320) structure.

9.156.2.5 unacknowledged_sample_count

```
DDS_Long DDS_ReliableWriterCacheChangedStatus::unacknowledged_sample_count
```

The current number of unacknowledged samples in the writer's cache.

A sample is considered unacknowledged if the writer has failed to receive an acknowledgement from one or more reliable readers matched to it.

9.156.2.6 unacknowledged_sample_count_peak

```
DDS_Long DDS_ReliableWriterCacheChangedStatus::unacknowledged_sample_count_peak
```

The highest value that unacknowledged_sample_count has reached until now.

9.156.2.7 replaced_unacknowledged_sample_count

```
DDS_LongLong DDS_ReliableWriterCacheChangedStatus::replaced_unacknowledged_sample_count
```

The number of unacknowledged samples that have been replaced in the writer's cache.

Total number of unacknowledged samples that have been replaced by a DataWriter after applying **DDS_KEEP_LAST↩ _HISTORY_QOS** (p. 604) policy.

9.157 DDS_ReliableWriterCacheEventCount Struct Reference

<<***extension***>> *(p. 432)* The number of times the number of unacknowledged samples in the cache of a reliable writer hit a certain well-defined threshold.

Public Attributes

- **DDS_Long total_count**

  The total number of times the event has occurred.
- **DDS_Long total_count_change**

  The incremental number of times the event has occurred since the listener was last invoked or the status read.

9.157.1 Detailed Description

<<***extension***>> *(p. 432)* The number of times the number of unacknowledged samples in the cache of a reliable writer hit a certain well-defined threshold.

The threshold interpretation depends on the usage of this data type in **DDS_ReliableWriterCacheChangedStatus** (p. 1302).

See also

DDS_ReliableWriterCacheChangedStatus (p. 1302)

9.157.2 Member Data Documentation

9.157.2.1 total_count

```
DDS_Long DDS_ReliableWriterCacheEventCount::total_count
```

The total number of times the event has occurred.

9.157.2.2 total_count_change

DDS_Long DDS_ReliableWriterCacheEventCount::total_count_change

The incremental number of times the event has occurred since the listener was last invoked or the status read.

9.158 DDS_RequestedDeadlineMissedStatus Struct Reference

DDS_REQUESTED_DEADLINE_MISSED_STATUS (p. 540)

Public Attributes

- **DDS_Long total_count**

  *Total cumulative count of the missed deadlines detected for any instance read by the **DDSDataReader** (p. 1556).*
- **DDS_Long total_count_change**

  The incremental number of missed deadlines detected since the last time the listener was called or the status was read.
- **DDS_InstanceHandle_t last_instance_handle**

  *Handle to the last instance in the **DDSDataReader** (p. 1556) for which a requested deadline was missed.*

9.158.1 Detailed Description

DDS_REQUESTED_DEADLINE_MISSED_STATUS (p. 540)

9.158.2 Member Data Documentation

9.158.2.1 total_count

DDS_Long DDS_RequestedDeadlineMissedStatus::total_count

Total cumulative count of the missed deadlines detected for any instance read by the **DDSDataReader** (p. 1556).

9.158.2.2 total_count_change

DDS_Long DDS_RequestedDeadlineMissedStatus::total_count_change

The incremental number of missed deadlines detected since the last time the listener was called or the status was read.

9.158.2.3 last_instance_handle

```
DDS_InstanceHandle_t DDS_RequestedDeadlineMissedStatus::last_instance_handle
```

Handle to the last instance in the **DDSDataReader** (p. 1556) for which a requested deadline was missed.

9.159 DDS_RequestedIncompatibleQosStatus Struct Reference

DDS_REQUESTED_INCOMPATIBLE_QOS_STATUS (p. 540)

Public Attributes

- **DDS_Long total_count**

  *Total cumulative count of how many times the concerned **DDSDataReader** (p. 1556) discovered a **DDSDataWriter** (p. 1589) for the same **DDSTopic** (p. 1885) with an offered QoS that is incompatible with that requested by the **DDSDataReader** (p. 1556).*
- **DDS_Long total_count_change**

  The change in `total_count` since the last time the listener was called or the status was read.
- **DDS_QosPolicyId_t last_policy_id**

  *The **DDS_QosPolicyId_t** (p. 557) of one of the policies that was found to be incompatible the last time an incompatibility was detected.*
- struct **DDS_QosPolicyCountSeq policies**

  *A list containing, for each policy, the total number of times that the concerned **DDSDataReader** (p. 1556) discovered a **DDSDataWriter** (p. 1589) for the same **DDSTopic** (p. 1885) with an offered QoS that is incompatible with that requested by the **DDSDataReader** (p. 1556).*

9.159.1 Detailed Description

DDS_REQUESTED_INCOMPATIBLE_QOS_STATUS (p. 540)

See also

DURABILITY (p. 596)
PRESENTATION (p. 615)
RELIABILITY (p. 631)
OWNERSHIP (p. 612)
LIVELINESS (p. 607)
DEADLINE (p. 583)
LATENCY_BUDGET (p. 605)
DESTINATION_ORDER (p. 584)

9.159.2 Member Data Documentation

9.159.2.1 total_count

DDS_Long DDS_RequestedIncompatibleQosStatus::total_count

Total cumulative count of how many times the concerned **DDSDataReader** (p. 1556) discovered a **DDSDataWriter** (p. 1589) for the same **DDSTopic** (p. 1885) with an offered QoS that is incompatible with that requested by the **DDSDataReader** (p. 1556).

9.159.2.2 total_count_change

DDS_Long DDS_RequestedIncompatibleQosStatus::total_count_change

The change in `total_count` since the last time the listener was called or the status was read.

9.159.2.3 last_policy_id

DDS_QosPolicyId_t DDS_RequestedIncompatibleQosStatus::last_policy_id

The **DDS_QosPolicyId_t** (p. 557) of one of the policies that was found to be incompatible the last time an incompatibility was detected.

9.159.2.4 policies

struct **DDS_QosPolicyCountSeq** DDS_RequestedIncompatibleQosStatus::policies

A list containing, for each policy, the total number of times that the concerned **DDSDataReader** (p. 1556) discovered a **DDSDataWriter** (p. 1589) for the same **DDSTopic** (p. 1885) with an offered QoS that is incompatible with that requested by the **DDSDataReader** (p. 1556).

9.160 DDS_ResourceLimitsQosPolicy Struct Reference

<<***spec***>> *(p. 209)* Controls the amount of physical memory allocated for DDS entities, if dynamic allocations are allowed, and how they occur. Also controls memory usage among different instance values for keyed topics.

Public Attributes

- **DDS_Long max_samples**

  *<<**basic**>> (p. 201) <<**spec**>> (p. 209) Represents the maximum samples the middleware can store for any one **DDSDataWriter** (p. 1589) (or **DDSDataReader** (p. 1556)).*
- **DDS_Long max_instances**

  *<<**basic**>> (p. 201) <<**spec**>> (p. 209) Represents the maximum number of instances a **DDSDataWriter** (p. 1589) (or **DDSDataReader** (p. 1556)) can manage.*
- **DDS_Long max_samples_per_instance**

  *<<**spec**>> (p. 209) Represents the maximum number of samples of any one instance a **DDSDataWriter** (p. 1589) (or **DDSDataReader** (p. 1556)) can manage.*
- **DDS_Long initial_samples**

  *<<**basic**>> (p. 201) <<**extension**>> (p. 213) Represents the initial samples the middleware will store for any one **DDSDataWriter** (p. 1589) (or **DDSDataReader** (p. 1556)).*
- **DDS_Long initial_instances**

  *<<**basic**>> (p. 201) <<**extension**>> (p. 213) Represents the initial number of instances a **DDSDataWriter** (p. 1589) (or **DDSDataReader** (p. 1556)) will manage.*
- **DDS_Long instance_hash_buckets**

  ***[DEPRECATED]** (p. **384**) <<**extension**>> (p. 213) Number of hash buckets for instances.*

9.160.1 Detailed Description

*<<**spec**>> (p. 209)* Controls the amount of physical memory allocated for DDS entities, if dynamic allocations are allowed, and how they occur. Also controls memory usage among different instance values for keyed topics.

Entity:

DDSTopic (p. 1885), **DDSDataReader** (p. 1556), **DDSDataWriter** (p. 1589)

Status:

DDS_SAMPLE_REJECTED_STATUS (p. 541), **DDS_SampleRejectedStatus** (p. 1356)

Properties:

RxO (p. 556) = NO
Changeable (p. 556) = **UNTIL ENABLE** (p. 556)

9.160.2 Usage

This policy controls the resources that RTI Connext can use to meet the requirements imposed by the application and other QoS settings.

For the reliability protocol (and **DDS_DurabilityQosPolicy** (p. 1005)), this QoS policy determines the actual maximum queue size when the **DDS_HistoryQosPolicy** (p. 1162) is set to **DDS_KEEP_ALL_HISTORY_QOS** (p. 604).

In general, this QoS policy is used to limit the amount of system memory that RTI Connext can allocate. For embedded real-time systems and safety-critical systems, pre-determination of maximum memory usage is often required. In addition, dynamic memory allocation could introduce non-deterministic latencies in time-critical paths.

This QoS policy can be set such that an entity does not dynamically allocate any more memory after its initialization phase.

If **DDSDataWriter** (p. 1589) objects are communicating samples faster than they are ultimately taken by the **DDSDataReader** (p. 1556) objects, the middleware will eventually hit against some of the QoS-imposed resource limits. Note that this may occur when just a single **DDSDataReader** (p. 1556) cannot keep up with its corresponding **DDSDataWriter** (p. 1589). The behavior in this case depends on the setting for the **RELIABILITY** (p. 631). If reliability is **DDS_BEST_EFFORT_RELIABILITY_QOS** (p. 633), then RTI Connext is allowed to drop samples. If the reliability is **DDS_RELIABLE_RELIABILITY_QOS** (p. 633), RTI Connext will block the **DDSDataWriter** (p. 1589) or discard the sample at the **DDSDataReader** (p. 1556) in order not to lose existing samples.

The constant **DDS_LENGTH_UNLIMITED** (p. 636) may be used to indicate the absence of a particular limit. For example setting **DDS_ResourceLimitsQosPolicy::max_samples_per_instance** (p. 1311) to **DDS_LENGTH_UNLIMITED** (p. 636) will cause RTI Connext not to enforce this particular limit.

If these resource limits are not set sufficiently, under certain circumstances the **DDSDataWriter** (p. 1589) may block on a write() call even though the **DDS_HistoryQosPolicy** (p. 1162) is **DDS_KEEP_LAST_HISTORY_QOS** (p. 604). To guarantee the writer does not block for **DDS_KEEP_LAST_HISTORY_QOS** (p. 604), make sure the resource limits are set such that:

```
max_samples >= max_instances * max_samples_per_instance
```

See also

DDS_ReliabilityQosPolicy (p. 1295)

DDS_HistoryQosPolicy (p. 1162)

9.160.3 Consistency

The setting of **DDS_ResourceLimitsQosPolicy::max_samples** (p. 1311) must be consistent with **DDS_ResourceLimitsQosPolicy::max_samples_per_instance** (p. 1311). For these two values to be consistent, it must be true that **DDS_ResourceLimitsQosPolicy::max_samples** (p. 1311) >= **DDS_ResourceLimitsQosPolicy::max_samples_per_instance** (p. 1311). As described above, this limit will not be enforced if **DDS_ResourceLimitsQosPolicy::max_samples_per_instance** (p. 1311) is set to **DDS_LENGTH_UNLIMITED** (p. 636).

The setting of **RESOURCE_LIMITS** (p. 635) `max_samples_per_instance` must be consistent with the **HISTORY** (p. 603) `depth`. For these two QoS to be consistent, it must be true that *depth* <= *max_samples_per_instance*.

See also

DDS_HistoryQosPolicy (p. 1162)

9.160.4 Member Data Documentation

9.160.4.1 max_samples

DDS_Long DDS_ResourceLimitsQosPolicy::max_samples

<<***basic***>> *(p. 201)* <<***spec***>> *(p. 209)* Represents the maximum samples the middleware can store for any one **DDSDataWriter** (p. 1589) (or **DDSDataReader** (p. 1556)).

Specifies the maximum number of data samples a **DDSDataWriter** (p. 1589) (or **DDSDataReader** (p. 1556)) can manage across all the instances associated with it.

For unkeyed types, this value has to be equal to max_samples_per_instance if max_samples_per_instance is not equal to **DDS_LENGTH_UNLIMITED** (p. 636).

When batching is enabled, the maximum number of data samples a **DDSDataWriter** (p. 1589) can manage will also be limited by **DDS_DataWriterResourceLimitsQosPolicy::max_batches** (p. 921).

[default] DDS_LENGTH_UNLIMITED (p. 636)

[range] [1, 100 million] or **DDS_LENGTH_UNLIMITED** (p. 636), >= initial_samples, >= max_samples_per_instance, >= **DDS_DataReaderResourceLimitsQosPolicy::max_samples_per_remote_writer** (p. 870) or >= **DDS_Rtps↩ ReliableWriterProtocol_t::heartbeats_per_max_samples** (p. 1327)

For **DDS_DataWriterQos** (p. 909) max_samples >= **DDS_RtpsReliableWriterProtocol_t::heartbeats_per_max_↩ samples** (p. 1327) in **DDS_DataWriterProtocolQosPolicy::rtps_reliable_writer** (p. 896) if batching is disabled.

Categories

Immutable (p. 333), **Durability** (p. 278), **Sample Memory Management** (p. 321), **Middleware Memory Management** (p. 297), **Data Availability** (p. 261)

9.160.4.2 max_instances

DDS_Long DDS_ResourceLimitsQosPolicy::max_instances

<<***basic***>> *(p. 201)* <<***spec***>> *(p. 209)* Represents the maximum number of instances a **DDSDataWriter** (p. 1589) (or **DDSDataReader** (p. 1556)) can manage.

[default] DDS_LENGTH_UNLIMITED (p. 636)

[range] [1, 1 million] or **DDS_LENGTH_UNLIMITED** (p. 636), >= initial_instances

Categories

Immutable (p. 333), **Durability** (p. 278), **Instance Lifecycle** (p. 286), **Sample Memory Management** (p. 321), **Middleware Memory Management** (p. 297), **Data Availability** (p. 261)

9.160.4.3 max_samples_per_instance

```
DDS_Long DDS_ResourceLimitsQosPolicy::max_samples_per_instance
```

<<***spec***>> *(p. 209)* Represents the maximum number of samples of any one instance a **DDSDataWriter** (p. 1589) (or **DDSDataReader** (p. 1556)) can manage.

While an unkeyed type is logically considered as a single instance, for unkeyed types this value has to be equal to max_samples or **DDS_LENGTH_UNLIMITED** (p. 636).

[default] DDS_LENGTH_UNLIMITED (p. 636)

[range] [1, 100 million] or **DDS_LENGTH_UNLIMITED** (p. 636), <= max_samples or **DDS_LENGTH_UNLIMITED** (p. 636), >= **DDS_HistoryQosPolicy::depth** (p. 1164)

Categories

Immutable (p. 333), **Durability** (p. 278), **Instance Lifecycle** (p. 286), **Sample Memory Management** (p. 321), **Middleware Memory Management** (p. 297), **Data Availability** (p. 261)

9.160.4.4 initial_samples

```
DDS_Long DDS_ResourceLimitsQosPolicy::initial_samples
```

<<***basic***>> (p. *201)* <<***extension***>> (p. *213)* Represents the initial samples the middleware will store for any one **DDSDataWriter** (p. 1589) (or **DDSDataReader** (p. 1556)).

Specifies the initial number of data samples a **DDSDataWriter** (p. 1589) (or **DDSDataReader** (p. 1556)) will manage across all the instances associated with it.

[default] 32

[range] [1,100 million], <= max_samples

Categories

Immutable (p. 333), **Sample Memory Management** (p. 321), **Middleware Memory Management** (p. 297)

9.160.4.5 initial_instances

```
DDS_Long DDS_ResourceLimitsQosPolicy::initial_instances
```

<<***basic***>> *(*p. *201)* <<***extension***>> *(*p. *213)* Represents the initial number of instances a **DDSDataWriter** (p. 1589) (or **DDSDataReader** (p. 1556)) will manage.

[default] 32

[range] [1,1 million], <= max_instances

Categories

Immutable (p. 333), **Instance Lifecycle** (p. 286), **Sample Memory Management** (p. 321), **Middleware Memory Management** (p. 297)

9.160.4.6 instance_hash_buckets

```
DDS_Long DDS_ResourceLimitsQosPolicy::instance_hash_buckets
```

[DEPRECATED] (p. **384)** <<***extension***>> *(*p. *213)* Number of hash buckets for instances.

The instance hash table facilitates instance lookup. A higher number of buckets decreases instance lookup time but increases the memory usage.

[default] 1 **[range]** [1,1 million]

Categories

Immutable (p. 333), **Instance Lifecycle** (p. 286)

9.161 DDS_RTPS_EntityId_t Struct Reference

From the DDS-RTPS specification: type used to hold the suffix part of the globally-unique RTPS-entity identifiers.

9.161.1 Detailed Description

From the DDS-RTPS specification: type used to hold the suffix part of the globally-unique RTPS-entity identifiers.

From DDS-RTPS Specification, clauses 8.4.2.1 and 9.3.1.

9.162 DDS_RTPS_GUID_t Struct Reference

From the DDS-RTPS specification: type used to hold a globally-unique RTPS-entity identifier.

9.162.1 Detailed Description

From the DDS-RTPS specification: type used to hold a globally-unique RTPS-entity identifier.

From DDS-RTPS Specification, clauses 8.4.2.1 and 9.3.1.

9.163 DDS_RtpsReliableReaderProtocol_t Struct Reference

<<***extension***>> *(*p. *213)* Qos related to reliable reader protocol defined in RTPS.

Public Attributes

- struct **DDS_Duration_t min_heartbeat_response_delay**

  <<***extension***>> *(*p. *213) The minimum delay to respond to a heartbeat.*
- struct **DDS_Duration_t max_heartbeat_response_delay**

  <<***extension***>> *(*p. *213) The maximum delay to respond to a heartbeat.*
- struct **DDS_Duration_t heartbeat_suppression_duration**

  <<***extension***>> *(*p. *213) The duration a reader ignores consecutively received heartbeats*
- struct **DDS_Duration_t nack_period**

  <<***extension***>> *(*p. *213) The period at which to send NACKs*
- **DDS_Long receive_window_size**

  <<***extension***>> *(*p. *213) The number of received out-of-order samples a reader can keep at a time*
- struct **DDS_Duration_t round_trip_time**

  <<***extension***>> *(*p. *213) The duration from sending a NACK to receiving a repair of a sample*
- struct **DDS_Duration_t app_ack_period**

  <<***extension***>> *(*p. *213) The period at which application-level acknowledgment messages are sent*
- struct **DDS_Duration_t min_app_ack_response_keep_duration**

  <<***extension***>> *(*p. *213) Minimum duration for which application-level acknowledgment response data is kept*
- **DDS_Long samples_per_app_ack**

  <<***extension***>> *(*p. *213) The minimum number of samples acknowledged by one application-level acknowledgment message*

9.163.1 Detailed Description

<<***extension***>> *(p. 213)* Qos related to reliable reader protocol defined in RTPS.

It is used to config reliable reader according to RTPS protocol.

Properties:

RxO (p. 556) = N/A
Changeable (p. 556) = **NO** (p. 556)

QoS:

DDS_DataReaderProtocolQosPolicy (p. 843) **DDS_DiscoveryConfigQosPolicy** (p. 934)

9.163.2 Member Data Documentation

9.163.2.1 min_heartbeat_response_delay

`struct` **`DDS_Duration_t`** `DDS_RtpsReliableReaderProtocol_t::min_heartbeat_response_delay`

<<***extension***>> *(p. 213)* The minimum delay to respond to a heartbeat.

When a reliable reader receives a heartbeat from a remote writer and finds out that it needs to send back an ACK/NACK message, the reader can choose to delay a while. This sets the value of the minimum delay.

[default] 0 seconds

[range] [0, 1 year], <= max_heartbeat_response_delay

Categories

Immutable (p. 333), **Reliability** (p. 314)

9.163.2.2 max_heartbeat_response_delay

`struct` **`DDS_Duration_t`** `DDS_RtpsReliableReaderProtocol_t::max_heartbeat_response_delay`

<<***extension***>> *(p. 213)* The maximum delay to respond to a heartbeat.

When a reliable reader receives a heartbeat from a remote writer and finds out that it needs to send back an ACK/NACK message, the reader can choose to delay a while. This sets the value of maximum delay.

[default] The default value depends on the container policy:

- For **DDS_DataReaderProtocolQosPolicy::rtps_reliable_reader** (p. 846): 0.5 seconds
- For **DDS_DiscoveryConfigQosPolicy::publication_reader** (p. 941): 0 seconds
- For **DDS_DiscoveryConfigQosPolicy::subscription_reader** (p. 942): 0 seconds
- For **DDS_DiscoveryConfigQosPolicy::participant_message_reader** (p. 947): 0 seconds

[range] [0, 1 year], >= min_heartbeat_response_delay

Categories

Immutable (p. 333), **Reliability** (p. 314)

9.163.2.3 heartbeat_suppression_duration

`struct` **`DDS_Duration_t`** `DDS_RtpsReliableReaderProtocol_t::heartbeat_suppression_duration`

<<***extension***>> *(p. 213)* The duration a reader ignores consecutively received heartbeats

When a reliable reader receives consecutive heartbeats within a short duration that will trigger redundant NACKs, the reader may ignore the latter heartbeat(s). This sets the duration during which additionally received heartbeats are suppressed.

[default] 0.0625 seconds

[range] [0, 1 year],

Categories

Immutable (p. 333), **Reliability** (p. 314)

9.163.2.4 nack_period

```
struct  DDS_Duration_t DDS_RtpsReliableReaderProtocol_t::nack_period
```

<<***extension***>> *(p. 213)* The period at which to send NACKs

A reliable reader will send periodic NACKs at this rate when it first matches with a reliable writer. The reader will stop sending NACKs when it has received all available historical data from the writer.

[default] 5 seconds

[range] [1 nanosec, 1 year]

Categories

Immutable (p. 333), **Reliability** (p. 314)

9.163.2.5 receive_window_size

```
DDS_Long DDS_RtpsReliableReaderProtocol_t::receive_window_size
```

<<***extension***>> *(p. 213)* The number of received out-of-order samples a reader can keep at a time

A reliable reader stores the out-of-order samples it receives until it can present them to the application in-order. The receive window is the maximum number of out-of-order samples that a reliable reader keeps at a given time. When the receive window is full, subsequently received out-of-order samples are dropped.

[default] 256

[range] [>= 1]

Categories

Immutable (p. 333), **Reliability** (p. 314)

9.163.2.6 round_trip_time

`struct` **`DDS_Duration_t`** `DDS_RtpsReliableReaderProtocol_t::round_trip_time`

<<***extension***>> *(p. 213)* The duration from sending a NACK to receiving a repair of a sample

This round-trip time is an estimate of the time starting from when the reader sends a NACK for a specific sample to when it receives that sample. For each sample, the reader will not send a subsequent NACK for it until the round-trip time has passed, thus preventing inefficient redundant requests.

[default] 0 seconds

[range] [0 nanosec, 1 year]

Categories

Immutable (p. 333), **Reliability** (p. 314)

9.163.2.7 app_ack_period

`struct` **`DDS_Duration_t`** `DDS_RtpsReliableReaderProtocol_t::app_ack_period`

<<***extension***>> *(p. 213)* The period at which application-level acknowledgment messages are sent

A **DDSDataReader** (p. 1556) sends application-level acknowledgment messages to a **DDSDataWriter** (p. 1589) at this periodic rate, and will continue sending until it receives a message from the **DDSDataWriter** (p. 1589) that it has received and processed the acknowledgment and an AppAckConfirmation has been received by the **DDSDataReader** (p. 1556). Note: application-level acknowledgment messages can also be sent non-periodically, as determined by **DDS_Rtps↩ ReliableReaderProtocol_t::samples_per_app_ack** (p. 1319).

[default] 5 seconds

[range] [1 nanosec, 1 year]

Categories

Immutable (p. 333), **Reliability** (p. 314), **Application Acknowledgement** (p. 253)

9.163.2.8 min_app_ack_response_keep_duration

```
struct DDS_Duration_t DDS_RtpsReliableReaderProtocol_t::min_app_ack_response_keep_duration
```

<<***extension***>> *(p. 213)* Minimum duration for which application-level acknowledgment response data is kept

The user-specified response data of an explicit application-level acknowledgment (called by **DDSDataReader::acknowledge_sample** (p. 1564) or **DDSDataReader::acknowledge_all** (p. 1565)) is cached by the **DDSDataReader** (p. 1556) for the purpose of reliably resending the data with the acknowledgment message. After this duration has passed from the time of the first acknowledgment, the response data is dropped from the cache and will not be resent with future acknowledgments for the corresponding sample(s).

[default] 0 sec

[range] [0 sec, 1 year]

Categories

Immutable (p. 333), **Reliability** (p. 314), **Application Acknowledgement** (p. 253)

9.163.2.9 samples_per_app_ack

```
DDS_Long DDS_RtpsReliableReaderProtocol_t::samples_per_app_ack
```

<<***extension***>> *(p. 213)* The minimum number of samples acknowledged by one application-level acknowledgment message

This setting applies only when **DDS_ReliabilityQosPolicy::acknowledgment_kind** (p. 1298) = **DDS_APPLICATION_EXPLICIT_ACKNOWLEDGMENT_MODE** (p. 634) or **DDS_APPLICATION_AUTO_ACKNOWLEDGMENT_MODE** (p. 633)

A **DDSDataReader** (p. 1556) will immediately send an application-level acknowledgment message when it has at least this many samples that have been acknowledged. It will not send an acknowledgment message until it has at least this many samples pending acknowledgment.

For example, calling **DDSDataReader::acknowledge_sample** (p. 1564) this many times consecutively will trigger the sending of an acknowledgment message. Calling **DDSDataReader::acknowledge_all** (p. 1565) may trigger the sending of an acknowledgment message, if at least this many samples are being acknowledged at once.

This is independent of the **DDS_RtpsReliableReaderProtocol_t::app_ack_period** (p. 1318), where a **DDSDataReader** (p. 1556) will send acknowledgement messages at the periodic rate regardless.

When this is set to **DDS_LENGTH_UNLIMITED** (p. 636), then acknowledgement messages are sent only periodically, at the rate set by **DDS_RtpsReliableReaderProtocol_t::app_ack_period** (p. 1318).

[default] 1

[range] [1, 1000000], or **DDS_LENGTH_UNLIMITED** (p. 636)

Categories

Immutable (p. 333), **Reliability** (p. 314), **Application Acknowledgement** (p. 253)

9.164 DDS_RtpsReliableWriterProtocol_t Struct Reference

<<***extension***>> *(*p. *213)* QoS related to the reliable writer protocol defined in RTPS.

Public Attributes

- **DDS_Long low_watermark**

  <<***extension***>> *(*p. *213) When the number of unacknowledged samples in the current send window of a reliable writer meets or falls below this threshold, the* ***DDS_RELIABLE_WRITER_CACHE_CHANGED_STATUS*** *(*p. *545) is considered to have changed.*
- **DDS_Long high_watermark**

  <<***extension***>> *(*p. *213) When the number of unacknowledged samples in the current send window of a reliable writer meets or exceeds this threshold, the* ***DDS_RELIABLE_WRITER_CACHE_CHANGED_STATUS*** *(*p. *545) is considered to have changed.*
- struct **DDS_Duration_t heartbeat_period**

  <<***basic***>> *(*p. *201)* <<***extension***>> *(*p. *213) The period at which to send heartbeats.*
- struct **DDS_Duration_t fast_heartbeat_period**

  <<***basic***>> *(*p. *201)* <<***extension***>> *(*p. *213) An alternative heartbeat period used when a reliable writer needs to flush its unacknowledged samples more quickly.*
- struct **DDS_Duration_t late_joiner_heartbeat_period**

  <<***basic***>> *(*p. *201)* <<***extension***>> *(*p. *213) An alternative heartbeat period used when a reliable reader joins late and needs to be caught up on cached samples of a reliable writer more quickly than the normal heartbeat rate.*
- struct **DDS_Duration_t virtual_heartbeat_period**

  <<***extension***>> *(*p. *213) The period at which to send virtual heartbeats. Virtual heartbeats inform the reliable reader about the range of samples currently present, for each virtual GUID, in the reliable writer's queue.*
- **DDS_Long samples_per_virtual_heartbeat**

  <<***extension***>> *(*p. *213) The number of samples that a reliable writer has to publish before sending a virtual heartbeat.*
- **DDS_Long max_heartbeat_retries**

  <<***extension***>> *(*p. *213) The maximum number of periodic heartbeat retries before marking a remote reader as inactive.*
- **DDS_Boolean inactivate_nonprogressing_readers**

  <<***extension***>> *(*p. *213) Whether to treat remote readers as inactive when their NACKs do not progress*
- **DDS_Long heartbeats_per_max_samples**

  <<***extension***>> *(*p. *213) The number of piggyback heartbeats sent per max send window.*
- struct **DDS_Duration_t min_nack_response_delay**

  <<***extension***>> *(*p. *213) The minimum delay to respond to a NACK or NACK_FRAG.*
- struct **DDS_Duration_t max_nack_response_delay**

  <<***extension***>> *(*p. *213) The maximum delay to respond to a NACK or NACK_FRAG.*
- struct **DDS_Duration_t nack_suppression_duration**

  <<***extension***>> *(*p. *213) The duration for ignoring consecutive NACKs/NACK_FRAGs that may trigger redundant repairs.*
- **DDS_Long max_bytes_per_nack_response**

  <<***extension***>> *(*p. *213) The maximum total message size when resending rejected samples or data fragments.*
- struct **DDS_Duration_t disable_positive_acks_min_sample_keep_duration**

  <<***extension***>> *(*p. *213) The minimum duration a sample is queued for ACK-disabled readers.*
- struct **DDS_Duration_t disable_positive_acks_max_sample_keep_duration**

  <<***extension***>> *(*p. *213) The maximum duration a sample is queued for ACK-disabled readers.*
- **DDS_Boolean disable_positive_acks_enable_adaptive_sample_keep_duration**

[DEPRECATED] (p. 384) <<extension>> (p. 213) Enables dynamic adjustment of sample keep duration in response to congestion.

- **DDS_Long disable_positive_acks_decrease_sample_keep_duration_factor**

  [DEPRECATED] (p. 384) <<extension>> (p. 213) Controls rate of contraction of dynamic sample keep duration.
- **DDS_Long disable_positive_acks_increase_sample_keep_duration_factor**

  [DEPRECATED] (p. 384) <<extension>> (p. 213) Controls rate of growth of dynamic sample keep duration.
- **DDS_Long min_send_window_size**

  <<basic>> (p. 201) <<extension>> (p. 213) Minimum size of send window of unacknowledged samples
- **DDS_Long max_send_window_size**

  <<basic>> (p. 201) <<extension>> (p. 213) Maximum size of send window of unacknowledged samples
- struct **DDS_Duration_t send_window_update_period**

  <<extension>> (p. 213) Period in which send window may be dynamically changed
- **DDS_Long send_window_increase_factor**

  <<extension>> (p. 213) Increases send window size by this percentage when reacting dynamically to network conditions
- **DDS_Long send_window_decrease_factor**

  <<extension>> (p. 213) Decreases send window size by this percentage when reacting dynamically to network conditions
- **DDS_Boolean enable_multicast_periodic_heartbeat**

  <<extension>> (p. 213) Whether periodic heartbeat messages are sent over multicast
- **DDS_Long multicast_resend_threshold**

  <<extension>> (p. 213) The minimum number of requesting readers needed to trigger a multicast resend
- **DDS_Boolean disable_repair_piggyback_heartbeat**

  <<extension>> (p. 213) Prevents piggyback heartbeats from being sent with repair samples

9.164.1 Detailed Description

<<extension>> (p. 213) QoS related to the reliable writer protocol defined in RTPS.

It is used to configure a reliable writer according to RTPS protocol.

The reliability protocol settings are applied to batches instead of individual data samples when batching is enabled.

Properties:

RxO (p. 556) = N/A
Changeable (p. 556) = **NO** (p. 556)

QoS:

DDS_DataWriterProtocolQosPolicy (p. 892) **DDS_DiscoveryConfigQosPolicy** (p. 934)

9.164.2 Member Data Documentation

9.164.2.1 low_watermark

DDS_Long DDS_RtpsReliableWriterProtocol_t::low_watermark

<<***extension***>> *(p. 213)* When the number of unacknowledged samples in the current send window of a reliable writer meets or falls below this threshold, the **DDS_RELIABLE_WRITER_CACHE_CHANGED_STATUS** (p. 545) is considered to have changed.

This value is measured in units of samples, except with batching configurations where it is measured in units of batches.

The value must be greater than or equal to zero and strictly less than high_watermark.

The high and low watermarks are used for switching between the regular and fast heartbeat rates (**DDS_RtpsReliable↩ WriterProtocol_t::heartbeat_period** (p. 1323) and **DDS_RtpsReliableWriterProtocol_t::fast_heartbeat_period** (p. 1323), respectively). When the number of unacknowledged samples in the queue of a reliable **DDSDataWriter** (p. 1589) meets or exceeds high_watermark, the **DDS_RELIABLE_WRITER_CACHE_CHANGED_STATUS** (p. 545) is changed, and the DataWriter will start heartbeating at **DDS_RtpsReliableWriterProtocol_t::fast_heartbeat_period** (p. 1323). When the number of samples meets or falls below low_watermark, **DDS_RELIABLE_WRITER_CACHE_↩ CHANGED_STATUS** (p. 545) is changed, and the heartbeat rate will return to the "normal" rate (**DDS_RtpsReliable↩ WriterProtocol_t::heartbeat_period** (p. 1323)).

[default] 0

[range] [0, 100 million], < high_watermark

Categories

Immutable (p. 333), **Reliability** (p. 314)

9.164.2.2 high_watermark

DDS_Long DDS_RtpsReliableWriterProtocol_t::high_watermark

<<***extension***>> *(p. 213)* When the number of unacknowledged samples in the current send window of a reliable writer meets or exceeds this threshold, the **DDS_RELIABLE_WRITER_CACHE_CHANGED_STATUS** (p. 545) is considered to have changed.

This value is measured in units of samples, except with batching configurations where it is measured in units of batches.

The value must be strictly greater than low_watermark and less than or equal to a maximum that depends on the container QoS policy:

In **DDS_DomainParticipantQos::discovery_config** (p. 975):

For **DDS_DiscoveryConfigQosPolicy::publication_writer** (p. 943)
high_watermark<= **DDS_AllocationSettings_t::max_count** (p. 797) in **DDS_DomainParticipantResourceLimits↩ QosPolicy::local_writer_allocation** (p. 981)

For **DDS_DiscoveryConfigQosPolicy::subscription_writer** (p. 945)
high_watermark<= **DDS_AllocationSettings_t::max_count** (p. 797) in **DDS_DomainParticipantResourceLimits↩ QosPolicy::local_reader_allocation** (p. 981)

In **DDS_DataWriterQos::protocol** (p. 915):

For **DDS_DataWriterProtocolQosPolicy::rtps_reliable_writer** (p. 896),

high_watermark<=**DDS_ResourceLimitsQosPolicy::max_samples** (p. 1311) if batching is disabled. Otherwise, high_watermark<=**DDS_DataWriterResourceLimitsQosPolicy::max_batches** (p. 921) high_watermark<=**DDS_↩ RtpsReliableWriterProtocol_t::max_send_window_size** (p. 1333)

[default] 1

[range] [1, 100 million] or **DDS_LENGTH_UNLIMITED** (p. 636), > low_watermark <= *maximum* which depends on the container policy

Categories

Immutable (p. 333), **Reliability** (p. 314)

9.164.2.3 heartbeat_period

`struct` **`DDS_Duration_t`** `DDS_RtpsReliableWriterProtocol_t::heartbeat_period`

<<***basic***>> *(p. 201)* <<***extension***>> *(p. 213)* The period at which to send heartbeats.

A reliable writer will send periodic heartbeats at this rate.

[default] The default value depends on the container policy:

- For **DDS_DataWriterProtocolQosPolicy::rtps_reliable_writer** (p. 896): 200 ms
- For **DDS_DiscoveryConfigQosPolicy::publication_writer** (p. 943): 3.0 seconds
- For **DDS_DiscoveryConfigQosPolicy::subscription_writer** (p. 945): 3.0 seconds
- For **DDS_DiscoveryConfigQosPolicy::participant_message_writer** (p. 947): 1.0 seconds

[range] [1 nanosec,1 year], >= **DDS_RtpsReliableWriterProtocol_t::fast_heartbeat_period** (p. 1323), >= **DDS_↩ RtpsReliableWriterProtocol_t::late_joiner_heartbeat_period** (p. 1324)

Categories

Immutable (p. 333), **Reliability** (p. 314)

9.164.2.4 fast_heartbeat_period

`struct` **DDS_Duration_t** `DDS_RtpsReliableWriterProtocol_t::fast_heartbeat_period`

*<<**basic**>> (p. 201) <<**extension**>> (p. 213)* An alternative heartbeat period used when a reliable writer needs to flush its unacknowledged samples more quickly.

This heartbeat period will be used when the number of unacknowledged samples in the cache of a reliable writer meets or exceeds the writer's high watermark and has not subsequently dropped to the low watermark. The normal period will be used at all other times.

This period must not be slower (i.e. must be of the same or shorter duration) than the normal heartbeat period.

[default] The default value depends on the container policy:

- For **DDS_DataWriterProtocolQosPolicy::rtps_reliable_writer** (p. 896): 200 ms seconds
- For **DDS_DiscoveryConfigQosPolicy::publication_writer** (p. 943): 3.0 seconds
- For **DDS_DiscoveryConfigQosPolicy::subscription_writer** (p. 945): 3.0 seconds
- For **DDS_DiscoveryConfigQosPolicy::participant_message_writer** (p. 947): 1.0 seconds

[range] [1 nanosec,1 year], <= **DDS_RtpsReliableWriterProtocol_t::heartbeat_period** (p. 1323)

Categories

Immutable (p. 333), **Reliability** (p. 314)

9.164.2.5 late_joiner_heartbeat_period

`struct` **DDS_Duration_t** `DDS_RtpsReliableWriterProtocol_t::late_joiner_heartbeat_period`

*<<**basic**>> (p. 201) <<**extension**>> (p. 213)* An alternative heartbeat period used when a reliable reader joins late and needs to be caught up on cached samples of a reliable writer more quickly than the normal heartbeat rate.

This heartbeat period will be used when a reliable reader joins after a reliable writer with non-volatile durability has begun publishing samples. Once the reliable reader has received all cached samples, it will be serviced at the same rate as other reliable readers.

This period must not be slower (i.e., must be of the same or shorter duration) than the normal heartbeat period.

A reliable writer will use whichever heartbeat period is faster, the current heartbeat period being used for other reliable readers or the **DDS_RtpsReliableWriterProtocol_t::late_joiner_heartbeat_period** (p. 1324), to service the late joining reader. This means that if the **DDS_RtpsReliableWriterProtocol_t::fast_heartbeat_period** (p. 1323) is currently being used and is faster than the late_joiner_heartbeat_period, then the fast_heartbeat_period will continue to be used for the late joiner as well.

[default] The default value depends on the container policy:

- For **DDS_DataWriterProtocolQosPolicy::rtps_reliable_writer** (p. 896): 200 seconds
- For **DDS_DiscoveryConfigQosPolicy::publication_writer** (p. 943): 3.0 seconds
- For **DDS_DiscoveryConfigQosPolicy::subscription_writer** (p. 945): 3.0 seconds
- For **DDS_DiscoveryConfigQosPolicy::participant_message_writer** (p. 947): 1.0 seconds

[range] [1 nanosec,1 year], <= **DDS_RtpsReliableWriterProtocol_t::heartbeat_period** (p. 1323)

Categories

Immutable (p. 333), **Reliability** (p. 314)

9.164.2.6 virtual_heartbeat_period

`struct` **`DDS_Duration_t`** `DDS_RtpsReliableWriterProtocol_t::virtual_heartbeat_period`

<<***extension***>> *(p. 213)* The period at which to send virtual heartbeats. Virtual heartbeats inform the reliable reader about the range of samples currently present, for each virtual GUID, in the reliable writer's queue.

A reliable writer will send periodic virtual heartbeats at this rate.

[default] The default value depends on the container policy:

- For **DDS_DataWriterProtocolQosPolicy::rtps_reliable_writer** (p. 896): **DDS_DURATION_AUTO** (p. 523). If **DDS_PresentationQosPolicy::access_scope** (p. 1252) is set to **DDS_GROUP_PRESENTATION_QOS** (p. 616) on the DataWriter, this value is set to **DDS_RtpsReliableWriterProtocol_t::heartbeat_period** (p. 1323). Otherwise, the value is set to **DDS_DURATION_INFINITE** (p. 522).
- For **DDS_DiscoveryConfigQosPolicy::publication_writer** (p. 943): **DDS_DURATION_INFINITE** (p. 522)
- For **DDS_DiscoveryConfigQosPolicy::subscription_writer** (p. 945): **DDS_DURATION_INFINITE** (p. 522)
- For **DDS_DiscoveryConfigQosPolicy::participant_message_writer** (p. 947): **DDS_DURATION_INFINITE** (p. 522)

[range] > 1 nanosec, **DDS_DURATION_INFINITE** (p. 522), or **DDS_DURATION_AUTO** (p. 523)

Categories

Immutable (p. 333), **Reliability** (p. 314)

9.164.2.7 samples_per_virtual_heartbeat

```
DDS_Long DDS_RtpsReliableWriterProtocol_t::samples_per_virtual_heartbeat
```

<<***extension***>> *(p. 213)* The number of samples that a reliable writer has to publish before sending a virtual heartbeat.

[default] DDS_LENGTH_UNLIMITED (p. 636)

[range] [1,1000000], **DDS_LENGTH_UNLIMITED** (p. 636)

Categories

Immutable (p. 333), **Reliability** (p. 314)

9.164.2.8 max_heartbeat_retries

```
DDS_Long DDS_RtpsReliableWriterProtocol_t::max_heartbeat_retries
```

<<***extension***>> *(p. 213)* The maximum number of *periodic* heartbeat retries before marking a remote reader as inactive.

When a remote reader has not acked all the samples the reliable writer has in its queue, and max_heartbeat_retries number of periodic heartbeats has been sent without receiving any ack/nack back, the remote reader will be marked as inactive (not alive) and be ignored until it resumes sending ack/nack.

Note that piggyback heartbeats do NOT count towards this value.

[default] 150

[range] [1, 1 million] or **DDS_LENGTH_UNLIMITED** (p. 636)

Categories

Immutable (p. 333), **Reliability** (p. 314), **Liveliness** (p. 292)

9.164.2.9 inactivate_nonprogressing_readers

DDS_Boolean DDS_RtpsReliableWriterProtocol_t::inactivate_nonprogressing_readers

<<***extension***>> *(p. 213)* Whether to treat remote readers as inactive when their NACKs do not progress

Nominally, a remote reader is marked inactive when a successive number of periodic heartbeats equal or greater than **DDS_RtpsReliableWriterProtocol_t::max_heartbeat_retries** (p. 1326) have been sent without receiving any ack/nacks back.

By setting this **DDS_BOOLEAN_TRUE** (p. 513), it changes the conditions of inactivating a remote reader: a reader will be considered inactive when it either does not send any ack/nacks or keeps sending non-progressing nacks for **DDS_RtpsReliableWriterProtocol_t::max_heartbeat_retries** (p. 1326) number of heartbeat periods, where a non-progressing nack is one whose oldest sample requested has not advanced from the oldest sample requested of the previous nack.

[default] DDS_BOOLEAN_FALSE (p. 513)

Categories

Immutable (p. 333), **Reliability** (p. 314), **Liveliness** (p. 292)

9.164.2.10 heartbeats_per_max_samples

DDS_Long DDS_RtpsReliableWriterProtocol_t::heartbeats_per_max_samples

<<***extension***>> *(p. 213)* The number of piggyback heartbeats sent per max send window.

When a DataWriter is configured with a fixed send window size (**DDS_RtpsReliableWriterProtocol_t::min_send_↩ window_size** (p. 1333) is equal to effective max_send_window_size), a piggyback heartbeat is sent every [(effective max send window size/heartbeats_per_max_samples)] number of samples written.

Otherwise, the number of piggyback heartbeats sent is scaled according to the current size of the send window. For example, consider a **DDS_RtpsReliableWriterProtocol_t::heartbeats_per_max_samples** (p. 1327) of 50. If the current send window size is 100, a piggyback heartbeat will be sent every 2 samples. If the send window size grows to 150, a piggyback heartbeat will be sent every 3 samples, and so on. Additionally, when the send window size grows, a piggyback heartbeat is sent with the next sample. (If it weren't, the sending of that heartbeat could be delayed, since the heartbeat rate scales with the increasing window size.)

The effective max send window is calculated as follows:

Without batching:

min (**DDS_ResourceLimitsQosPolicy::max_samples** (p. 1311), **DDS_RtpsReliableWriterProtocol_t::max_send↩ _window_size** (p. 1333))

With batching:

min (**DDS_DataWriterResourceLimitsQosPolicy::max_batches** (p. 921), **DDS_RtpsReliableWriterProtocol_t↩ ::max_send_window_size** (p. 1333))

If heartbeats_per_max_samples is set to zero, no piggyback heartbeats will be sent.

If current send window size is **DDS_LENGTH_UNLIMITED** (p. 636), 100 million is assumed as the effective max send window.

[default] The default value depends on the container policy:

- For **DDS_DataWriterProtocolQosPolicy::rtps_reliable_writer** (p. 896): 8
- For **DDS_DiscoveryConfigQosPolicy::publication_writer** (p. 943): 8
- For **DDS_DiscoveryConfigQosPolicy::subscription_writer** (p. 945): 8
- For **DDS_DiscoveryConfigQosPolicy::participant_message_writer** (p. 947): 1

[range] [0, 100 million]

- For **DDS_DiscoveryConfigQosPolicy::publication_writer** (p. 943): heartbeats_per_max_samples<= **DDS_AllocationSettings_t::max_count** (p. 797) in **DDS_DomainParticipantResourceLimitsQosPolicy::local_writer_allocation** (p. 981)
- For **DDS_DiscoveryConfigQosPolicy::subscription_writer** (p. 945): heartbeats_per_max_samples<= **DDS_AllocationSettings_t::max_count** (p. 797) in **DDS_DomainParticipantResourceLimitsQosPolicy::local_reader_allocation** (p. 981)
- For **DDS_DataWriterProtocolQosPolicy::rtps_reliable_writer** (p. 896):

heartbeats_per_max_samples<= **DDS_ResourceLimitsQosPolicy::max_samples** (p. 1311) if batching is disabled. Otherwise:
heartbeats_per_max_samples<= **DDS_DataWriterResourceLimitsQosPolicy::max_batches** (p. 921)

heartbeats_per_max_samples<= **DDS_RtpsReliableWriterProtocol_t::max_send_window_size** (p. 1333)

Categories

Immutable (p. 333), **Reliability** (p. 314)

9.164.2.11 min_nack_response_delay

`struct` **`DDS_Duration_t`** `DDS_RtpsReliableWriterProtocol_t::min_nack_response_delay`

<<***extension***>> *(p. 213)* The minimum delay to respond to a NACK or NACK_FRAG.

When a reliable writer receives a NACK or a NACK_FRAG from a remote reader, the writer can choose to delay a while before it sends repair samples/fragments or a heartbeat. This sets the value of the minimum delay.

Delaying repairs can lower the amount of duplicate repair data sent to the remote reader. If the writer receives the same NACKs or NACK_FRAGs back-to-back within this wait time, it will combine them into one repair action.

[default] 0 seconds

[range] [0, 1 day], <= max_nack_response_delay

Categories

Immutable (p. 333), **Reliability** (p. 314), **Fragmentation** (p. 284)

9.164.2.12 max_nack_response_delay

`struct DDS_Duration_t DDS_RtpsReliableWriterProtocol_t::max_nack_response_delay`

<<***extension***>> *(p. 213)* The maximum delay to respond to a NACK or NACK_FRAG.

This sets the value of maximum delay between receiving a NACK or a NACK_FRAG and sending repair samples/fragments or a heartbeat.

[default] The default value depends on the container policy:

- For **DDS_DataWriterProtocolQosPolicy::rtps_reliable_writer** (p. 896): 0.2 seconds
- For **DDS_DiscoveryConfigQosPolicy::publication_writer** (p. 943): 0 seconds
- For **DDS_DiscoveryConfigQosPolicy::subscription_writer** (p. 945): 0 seconds
- For **DDS_DiscoveryConfigQosPolicy::participant_message_writer** (p. 947): 0 seconds

[range] [0,1 day], >= min_nack_response_delay

Categories

Immutable (p. 333), **Reliability** (p. 314), **Fragmentation** (p. 284)

9.164.2.13 nack_suppression_duration

`struct DDS_Duration_t DDS_RtpsReliableWriterProtocol_t::nack_suppression_duration`

<<***extension***>> *(p. 213)* The duration for ignoring consecutive NACKs/NACK_FRAGs that may trigger redundant repairs.

A reliable writer may receive consecutive NACKs/NACK_FRAGs within a short duration from a remote reader that will trigger the sending of redundant repair messages.

This specifies the duration during which consecutive NACKs/NACK_FRAGs are ignored to prevent redundant repairs from being sent.

A redundant NACK is a NACK that requests the same or lower first sequence number (SN) than the last NACK from the same reader.

A redundant NACK_FRAG is a NACK_FRAG that requests the same first fragment number for a sample as the last NACK_FRAG from the same reader.

The suppression duration applies separately to NACKs and NACK_FRAGS. For example, replying to a NACK_FRAG message in between receiving two NACK messages will not cause the second NACK message to be suppressed if the two NACK messages are received more than the suppression duration apart.

[default] 0 seconds

[range] [0, 1 day]

Categories

Immutable (p. 333), **Reliability** (p. 314), **Fragmentation** (p. 284)

9.164.2.14 max_bytes_per_nack_response

```
DDS_Long DDS_RtpsReliableWriterProtocol_t::max_bytes_per_nack_response
```

<<***extension***>> *(p. 213)* The maximum total message size when resending rejected samples or data fragments.

As part of the reliable communication protocol, DataWriters send heartbeat (HB) messages to their DataReaders. Each HB message contains the sequence number of the most recent sample sent by the DataWriter.

In response, a DataReader may send a negative acknowledgement (NACK) message, indicating what sequence numbers it did not receive, if any. If the samples were fragmented, it can also send a NACK_FRAG requesting the fragments of a sample that did not arrive. If the DataReader is missing some samples or fragments, the DataWriter will send them again.

max_bytes_per_nack_response determines the maximum size of the message sent by the DataWriter in response to a NACK or NACK_FRAG. This message may contain multiple samples or fragments. The DataWriter will always send at least one message, even if the size of that message exceeds the max_bytes_per_nack_response value.

If the total size of all samples or fragments that needs to be resent is larger than max_bytes_per_nack_response, the remaining samples or fragments will be resent the next time a NACK or NACK_FRAG arrives.

Note: If **DDS_RtpsReliableWriterProtocol_t::min_nack_response_delay** (p. 1328) and **DDS_RtpsReliableWriter↩ Protocol_t::max_nack_response_delay** (p. 1328) are both **DDS_DURATION_ZERO** (p. 523) (which means repairs are processed immediately), this limit applies separately to NACKs and NACK_FRAGs: repairing a NACK in this case does not affect the maximum size of a NACK_FRAG response and vice versa. Otherwise, the limit applies to the total size of the response, including received NACKs and NACK_FRAGs.

[default] The default value depends on the container policy:

- For **DDS_DataWriterProtocolQosPolicy::rtps_reliable_writer** (p. 896): 131072 bytes
- For **DDS_DiscoveryConfigQosPolicy::publication_writer** (p. 943): 131072 bytes
- For **DDS_DiscoveryConfigQosPolicy::subscription_writer** (p. 945): 131072 bytes
- For **DDS_DiscoveryConfigQosPolicy::participant_message_writer** (p. 947): 9216 bytes

[range] [0, 1 GB]

Categories

Immutable (p. 333), **Reliability** (p. 314), **Flow Control** (p. 282), **Fragmentation** (p. 284)

9.164.2.15 disable_positive_acks_min_sample_keep_duration

`struct` **`DDS_Duration_t`** `DDS_RtpsReliableWriterProtocol_t::disable_positive_acks_min_sample_keep_↩ duration`

<<***extension***>> *(p. 213)* The minimum duration a sample is queued for ACK-disabled readers.

When positive ACKs are disabled for a data writer (**DDS_DataWriterProtocolQosPolicy::disable_positive_acks** (p. 894) = **DDS_BOOLEAN_TRUE** (p. 513)) or a data reader (**DDS_DataReaderProtocolQosPolicy::disable_↩ positive_acks** (p. 845) = **DDS_BOOLEAN_TRUE** (p. 513)), a sample is available from the data writer's queue for at least this duration, after which the sample may be considered to be acknowledged.

[default] 1 millisecond

[range] [0,1 year], <= **DDS_RtpsReliableWriterProtocol_t::disable_positive_acks_max_sample_keep_duration** (p. 1331)

Categories

Immutable (p. 333), **Reliability** (p. 314), **Durability** (p. 278)

9.164.2.16 disable_positive_acks_max_sample_keep_duration

`struct` **`DDS_Duration_t`** `DDS_RtpsReliableWriterProtocol_t::disable_positive_acks_max_sample_keep_↩ duration`

<<***extension***>> *(p. 213)* The maximum duration a sample is queued for ACK-disabled readers.

When positive ACKs are disabled for a data writer (**DDS_DataWriterProtocolQosPolicy::disable_positive_acks** (p. 894) = **DDS_BOOLEAN_TRUE** (p. 513)) or a data reader (**DDS_DataReaderProtocolQosPolicy::disable_↩ positive_acks** (p. 845) = **DDS_BOOLEAN_TRUE** (p. 513)), a sample is available from the data writer's queue for at most this duration, after which the sample is considered to be acknowledged.

[default] 1 second

[range] [0,1 year], >= **DDS_RtpsReliableWriterProtocol_t::disable_positive_acks_min_sample_keep_duration** (p. 1330)

Categories

Immutable (p. 333), **Reliability** (p. 314), **Durability** (p. 278)

9.164.2.17 disable_positive_acks_enable_adaptive_sample_keep_duration

DDS_Boolean DDS_RtpsReliableWriterProtocol_t::disable_positive_acks_enable_adaptive_sample_keep_duration

[DEPRECATED] (p. **384)** <<***extension***>> *(*p. *213)* Enables dynamic adjustment of sample keep duration in response to congestion.

For dynamic networks where a static minimum sample keep duration may not provide sufficient performance or reliability, setting **DDS_RtpsReliableWriterProtocol_t::disable_positive_acks_enable_adaptive_sample_keep_duration** (p. 1331) = **DDS_BOOLEAN_TRUE** (p. 513), enables the sample keep duration to be dynamically adjusted to adapt to network conditions. The keep duration changes according to the detected level of congestion, which is determined to be proportional to the rate of NACKs received. An adaptive algorithm automatically controls the keep duration to optimize throughput and reliability.

To relieve high congestion, the keep duration is increased to effectively decrease the send rate; this lengthening of the keep duration is controlled by **DDS_RtpsReliableWriterProtocol_t::disable_positive_acks_increase_sample_keep_duration_factor** (p. 1332). Alternatively, when congestion is low, the keep duration is decreased to effectively increase send rate; this shortening of the keep duration is controlled by **DDS_RtpsReliableWriterProtocol_t::disable_positive_acks_decrease_sample_keep_duration_factor** (p. 1332).

The lower and upper bounds of the dynamic sample keep duration are set by **DDS_RtpsReliableWriterProtocol_t::disable_positive_acks_min_sample_keep_duration** (p. 1330) and **DDS_RtpsReliableWriterProtocol_t::disable_positive_acks_max_sample_keep_duration** (p. 1331), respectively.

When **DDS_RtpsReliableWriterProtocol_t::disable_positive_acks_enable_adaptive_sample_keep_duration** (p. 1331) = **DDS_BOOLEAN_FALSE** (p. 513), the sample keep duration is set to **DDS_RtpsReliableWriterProtocol_t::disable_positive_acks_min_sample_keep_duration** (p. 1330) .

[default] DDS_BOOLEAN_TRUE (p. 513)

Categories

Immutable (p. 333), **Reliability** (p. 314), **Durability** (p. 278)

9.164.2.18 disable_positive_acks_decrease_sample_keep_duration_factor

DDS_Long DDS_RtpsReliableWriterProtocol_t::disable_positive_acks_decrease_sample_keep_duration_factor

[DEPRECATED] (p. **384)** <<***extension***>> *(*p. *213)* Controls rate of contraction of dynamic sample keep duration.

Used when **DDS_RtpsReliableWriterProtocol_t::disable_positive_acks_enable_adaptive_sample_keep_duration** (p. 1331) = **DDS_BOOLEAN_TRUE** (p. 513).

When the adaptive algorithm determines that the keep duration should be decreased, this factor (a percentage) is multiplied with the current keep duration to get the new shorter keep duration. For example, if the current keep duration is 20 milliseconds, using the default factor of 95% would result in a new keep duration of 19 milliseconds.

[default] 95

[range] <= 100

Categories

Immutable (p. 333), **Reliability** (p. 314), **Durability** (p. 278)

9.164.2.19 disable_positive_acks_increase_sample_keep_duration_factor

DDS_Long DDS_RtpsReliableWriterProtocol_t::disable_positive_acks_increase_sample_keep_duration_↩ factor

[DEPRECATED] (p. **384)** <<***extension***>> *(*p. *213)* Controls rate of growth of dynamic sample keep duration.

Used when **DDS_RtpsReliableWriterProtocol_t::disable_positive_acks_enable_adaptive_sample_keep_↩ duration** (p. 1331) = **DDS_BOOLEAN_TRUE** (p. 513).

When the adaptive algorithm determines that the keep duration should be increased, this factor (a percentage) is multiplied with the current keep duration to get the new longer keep duration. For example, if the current keep duration is 20 milliseconds, using the default factor of 150% would result in a new keep duration of 30 milliseconds.

[default] 150

[range] >= 100

Categories

Immutable (p. 333), **Reliability** (p. 314), **Durability** (p. 278)

9.164.2.20 min_send_window_size

DDS_Long DDS_RtpsReliableWriterProtocol_t::min_send_window_size

<<***basic***>> *(*p. *201)* <<***extension***>> *(*p. *213)* Minimum size of send window of unacknowledged samples

A **DDSDataWriter** (p. 1589) has a limit on the number of unacknowledged samples in-flight at a time. This send window can be configured to have a minimum size (this field) and a maximum size (max_send_window_size). The send window can dynamically change, between the min and max sizes, to throttle the effective send rate in response to changing network congestion, as measured by negative acknowledgements received.

When a variable sized send window is used (i.e., when min_send_window_size and max_send_window_size are not set to the same value) the send window is initialized to min_send_window_size.

[default] DDS_LENGTH_UNLIMITED (p. 636)

[range] > 0, <= max_send_window_size, or **DDS_LENGTH_UNLIMITED** (p. 636)

See also

DDS_RtpsReliableWriterProtocol_t::max_send_window_size (p. 1333)

DDS_RtpsReliableWriterProtocol_t::low_watermark (p. 1321)

DDS_RtpsReliableWriterProtocol_t::high_watermark (p. 1322)

DDS_ReliableWriterCacheChangedStatus::full_reliable_writer_cache (p. 1303)

Categories

Immutable (p. 333), **Reliability** (p. 314), **Flow Control** (p. 282)

9.164.2.21 max_send_window_size

```
DDS_Long DDS_RtpsReliableWriterProtocol_t::max_send_window_size
```

<<***basic***>> *(p. 201)* <<***extension***>> *(p. 213)* Maximum size of send window of unacknowledged samples

A **DDSDataWriter** (p. 1589) has a limit on the number of unacknowledged samples in-flight at a time. This send window can be configured to have a minimum size (min_send_window_size) and a maximum size (this field). The send window can dynamically change, between the min and max sizes, to throttle the effective send rate in response to changing network congestion, as measured by negative acknowledgements received.

When a variable sized send window is used (i.e., when min_send_window_size and max_send_window_size are not set to the same value) the send window is initialized to min_send_window_size.

When both min_send_window_size and max_send_window_size are **DDS_LENGTH_UNLIMITED** (p. 636), then either **DDS_ResourceLimitsQosPolicy::max_samples** (p. 1311) (for non-batching) or **DDS_DataWriterResource↩ LimitsQosPolicy::max_batches** (p. 921) (for batching) serves as the effective max_send_window_size. When **DDS↩ _ResourceLimitsQosPolicy::max_samples** (p. 1311) (for non-batching) or **DDS_DataWriterResourceLimitsQos↩ Policy::max_batches** (p. 921) (for batching) is less than max_send_window_size, then it serves as the effective max↩ _send_window_size. If it is also less than min_send_window_size, then effectively both min and max send window sizes are equal to max_samples or max_batches.

In addition, the low and high watermarks are scaled down linearly to stay within the current send window size, and the full reliable queue status is set when the send window is full.

[default] DDS_LENGTH_UNLIMITED (p. 636)

[range] $>$ 0, $>=$ min_send_window_size, or **DDS_LENGTH_UNLIMITED** (p. 636)

See also

DDS_RtpsReliableWriterProtocol_t::min_send_window_size (p. 1333)

DDS_RtpsReliableWriterProtocol_t::low_watermark (p. 1321)

DDS_RtpsReliableWriterProtocol_t::high_watermark (p. 1322)

DDS_ReliableWriterCacheChangedStatus::full_reliable_writer_cache (p. 1303)

Categories

Immutable (p. 333), **Reliability** (p. 314), **Flow Control** (p. 282)

9.164.2.22 send_window_update_period

```
struct DDS_Duration_t DDS_RtpsReliableWriterProtocol_t::send_window_update_period
```

<<***extension***>> *(p. 213)* Period in which send window may be dynamically changed

The **DDSDataWriter** (p. 1589)'s send window will dynamically change, between the min and max send window sizes, to throttle the effective send rate in response to changing network congestion, as measured by negative acknowledgements received.

The change in send window size happens at this update period, whereupon the send window is either increased or decreased in size according to the increase or decrease factors, respectively.

[default] The default value depends on the container policy:

- For **DDS_DataWriterProtocolQosPolicy::rtps_reliable_writer** (p. 896): 3 seconds
- For **DDS_DiscoveryConfigQosPolicy::publication_writer** (p. 943): 3 seconds
- For **DDS_DiscoveryConfigQosPolicy::subscription_writer** (p. 945): 3 seconds
- For **DDS_DiscoveryConfigQosPolicy::participant_message_writer** (p. 947): 1 second

[range] > [0,1 year]

See also

DDS_RtpsReliableWriterProtocol_t::send_window_increase_factor (p. 1335), **DDS_RtpsReliableWriter↩ Protocol_t::send_window_decrease_factor** (p. 1336)

Categories

Immutable (p. 333), **Reliability** (p. 314), **Flow Control** (p. 282)

9.164.2.23 send_window_increase_factor

DDS_Long DDS_RtpsReliableWriterProtocol_t::send_window_increase_factor

<<***extension***>> *(p. 213)* Increases send window size by this percentage when reacting dynamically to network conditions

The **DDSDataWriter** (p. 1589)'s send window will dynamically change, between the min and max send window sizes, to throttle the effective send rate in response to changing network congestion, as measured by negative acknowledgements received.

After an update period during which no negative acknowledgements were received, the send window will be increased by this factor. The factor is treated as a percentage, where a factor of 150 would increase the send window by 150%. The increased send window size will not exceed the max_send_window_size.

[default] 105

[range] $>$ 100

See also

DDS_RtpsReliableWriterProtocol_t::send_window_update_period (p. 1334), **DDS_RtpsReliableWriter↩ Protocol_t::send_window_decrease_factor** (p. 1336)

Categories

Immutable (p. 333), **Reliability** (p. 314), **Flow Control** (p. 282)

9.164.2.24 send_window_decrease_factor

DDS_Long DDS_RtpsReliableWriterProtocol_t::send_window_decrease_factor

<<***extension***>> *(p. 213)* Decreases send window size by this percentage when reacting dynamically to network conditions

The **DDSDataWriter** (p. 1589)'s send window will dynamically change, between the min and max send window sizes, to throttle the effective send rate in response to changing network congestion, as measured by negative acknowledgements received.

When increased network congestion causes a negative acknowledgement to be received by a writer, the send window will be decreased by this factor to throttle the effective send rate. The factor is treated as a percentage, where a factor of 80 would decrease the send window to 80% of its previous size. The decreased send window size will not be less than the min_send_window_size.

[default] The default value depends on the container policy:

- For **DDS_DataWriterProtocolQosPolicy::rtps_reliable_writer** (p. 896): 70
- For **DDS_DiscoveryConfigQosPolicy::publication_writer** (p. 943): 50
- For **DDS_DiscoveryConfigQosPolicy::subscription_writer** (p. 945): 50
- For **DDS_DiscoveryConfigQosPolicy::participant_message_writer** (p. 947): 50

[range] [0, 100]

See also

DDS_RtpsReliableWriterProtocol_t::send_window_update_period (p. 1334), **DDS_RtpsReliableWriter↩ Protocol_t::send_window_increase_factor** (p. 1335)

Categories

Immutable (p. 333), **Reliability** (p. 314), **Flow Control** (p. 282)

9.164.2.25 enable_multicast_periodic_heartbeat

```
DDS_Boolean DDS_RtpsReliableWriterProtocol_t::enable_multicast_periodic_heartbeat
```

<<***extension***>> *(p. 213)* Whether periodic heartbeat messages are sent over multicast

When enabled, if a reader has a multicast destination, then the writer will send its periodic HEARTBEAT messages to that destination. Otherwise, if not enabled or the reader does not have a multicast destination, the writer will send its periodic HEARTBEATs over unicast.

[default] DDS_BOOLEAN_FALSE (p. 513)

Categories

Immutable (p. 333), **Reliability** (p. 314), **Transports** (p. 327)

9.164.2.26 multicast_resend_threshold

DDS_Long DDS_RtpsReliableWriterProtocol_t::multicast_resend_threshold

<<***extension***>> *(p. 213)* The minimum number of requesting readers needed to trigger a multicast resend

Given readers with multicast destinations, when a reader NACKs for samples to be resent, the writer can either resend them over unicast or multicast. In order for the writer to resend over multicast, this threshold is the minimum number of readers of the same multicast group that the writer must receive NACKs from within a single response-delay. This allows the writer to coalesce near-simultaneous unicast resends into a multicast resend. Note that a threshold of 1 means that all resends will be sent over multicast, if available.

[default] 2

[range] [>= 1]

Categories

Immutable (p. 333), **Reliability** (p. 314), **Transports** (p. 327)

9.164.2.27 disable_repair_piggyback_heartbeat

DDS_Boolean DDS_RtpsReliableWriterProtocol_t::disable_repair_piggyback_heartbeat

<<***extension***>> *(p. 213)* Prevents piggyback heartbeats from being sent with repair samples

When samples are repaired, the **DDSDataWriter** (p. 1589) resends **DDS_RtpsReliableWriterProtocol_t::max_bytes_per_nack_response** (p. 1329) bytes and a piggyback heartbeat with each message. You can configure the **DDSDataWriter** (p. 1589) to not send the piggyback heartbeat and instead rely on the **DDS_RtpsReliableWriterProtocol_t::late_joiner_heartbeat_period** (p. 1324) to control the throughput used to repair samples. This field is mutable only for **DDS_DataWriterProtocolQosPolicy::rtps_reliable_writer** (p. 896). **[default] DDS_BOOLEAN_FALSE** (p. 513)

Categories

Immutable (p. 333), **Reliability** (p. 314), **Flow Control** (p. 282)

9.165 DDS_RtpsWellKnownPorts_t Struct Reference

<<***extension***>> *(p. 213)* RTPS well-known port mapping configuration.

Public Attributes

- **DDS_Long port_base**

  <<***extension***>> (p. *213) The base port offset.*
- **DDS_Long domain_id_gain**

  <<***extension***>> (p. *213) Tunable domain gain parameter.*
- **DDS_Long participant_id_gain**

  <<***extension***>> (p. *213) Tunable participant gain parameter.*
- **DDS_Long builtin_multicast_port_offset**

  <<***extension***>> (p. *213) Additional offset for **metatraffic** multicast port.*
- **DDS_Long builtin_unicast_port_offset**

  <<***extension***>> (p. *213) Additional offset for **metatraffic** unicast port.*
- **DDS_Long user_multicast_port_offset**

  <<***extension***>> (p. *213) Additional offset for **usertraffic** multicast port.*
- **DDS_Long user_unicast_port_offset**

  <<***extension***>> (p. *213) Additional offset for **usertraffic** unicast port.*

9.165.1 Detailed Description

<<***extension***>> *(p. 213)* RTPS well-known port mapping configuration.

RTI Connext uses the RTPS wire protocol. The discovery protocols defined by RTPS rely on well-known ports to initiate discovery. These well-known ports define the multicast and unicast ports on which a Participant will listen for discovery **metatraffic** from other Participants. The discovery metatraffic contains all the information required to establish the presence of remote DDS entities in the network.

The well-known ports are defined by RTPS in terms of port mapping expressions with several tunable parameters, which allow you to customize what network ports are used by RTI Connext. These parameters are exposed in **DDS_Rtps↩ WellKnownPorts_t** (p. 1338). In order for all Participants in a system to correctly discover each other, it is important that they all use the same port mapping expressions.

The actual port mapping expressions, as defined by the RTPS specification, can be found below. In addition to the parameters listed in **DDS_RtpsWellKnownPorts_t** (p. 1338), the port numbers depend on:

- `domain_id`, as specified in **DDSDomainParticipantFactory::create_participant** (p. 1710)
- `participant_id`, as specified using **DDS_WireProtocolQosPolicy::participant_id** (p. 1514)

The `domain_id` parameter ensures no port conflicts exist between Participants belonging to different domains. This also means that discovery metatraffic in one domain is not visible to Participants in a different domain. The `participant_id` parameter ensures that unique unicast port numbers are assigned to Participants belonging to the same domain on a given host.

The *metatraffic_unicast_port* is used to exchange discovery metatraffic using unicast.

```
metatraffic_unicast_port = port_base + (domain_id_gain * domain_id) + (participant_id_gain * participant_id) + b
```

The *metatraffic_multicast_port* is used to exchange discovery metatraffic using multicast. The corresponding multicast group addresses are specified via **DDS_DiscoveryQosPolicy::multicast_receive_addresses** (p. 962) on a **DDSDomainParticipant** (p. 1619) entity.

```
metatraffic_multicast_port = port_base + (domain_id_gain * domain_id) + builtin_multicast_port_offset
```

RTPS also defines the *default* multicast and unicast ports on which DataReaders and DataWriters receive **usertraffic**. These default ports can be overridden using the **DDS_DataReaderQos::multicast** (p. 864), **DDS_DataReaderQos↩ ::unicast** (p. 864), or by the **DDS_DataWriterQos::unicast** (p. 915) QoS policies.

The *usertraffic_unicast_port* is used to exchange user data using unicast.

```
usertraffic_unicast_port = port_base + (domain_id_gain * domain_id) + (participant_id_gain * participant_id) + u
```

The *usertraffic_multicast_port* is used to exchange user data using multicast. The corresponding multicast group addresses can be configured using **DDS_TransportMulticastQosPolicy** (p. 1415).

```
usertraffic_multicast_port = port_base + (domain_id_gain * domain_id) + user_multicast_port_offset
```

By default, the port mapping parameters are configured to compliant with OMG's DDS Interoperability Wire Protocol (see also **DDS_INTEROPERABLE_RTPS_WELL_KNOWN_PORTS** (p. 659)).

The OMG's DDS Interoperability Wire Protocol compliant port mapping parameters are *not* backwards compatible with previous versions of the RTI Connext middleware.

When modifying the port mapping parameters, care must be taken to avoid port aliasing. This would result in undefined discovery behavior. The chosen parameter values will also determine the maximum possible number of domains in the system and the maximum number of participants per domain. Additionally, any resulting mapped port number must be within the range imposed by the underlying transport. For example, for UDPv4, this range typically equals [1024 - 65535].

Note: On Windows, you should avoid using ports 49152 through 65535 for inbound traffic. RTI Connext's ephemeral ports (see "Ports Used for Communication" in the `User's Manual`) may be within that range (see `https↩ ://msdn.microsoft.com/en-us/library/windows/desktop/ms737550(v=vs.85).aspx`). With the default RtpsWellKnownPorts settings, port 49152 corresponds to domain ID 167, so using domain IDs 168 through 232 on Windows introduces the risk of a port collision and failure to create the Domain Participant when using multicast discovery. You may see this error:

RTIOsapiSocket_bindWithIP:OS bind() failure, error 0X271D: An attempt was made to access a socket in a way forbidden by its access permissions.

QoS:

DDS_WireProtocolQosPolicy (p. 1511)

9.165.2 Member Data Documentation

9.165.2.1 port_base

```
DDS_Long DDS_RtpsWellKnownPorts_t::port_base
```

<<***extension***>> *(p. 213)* The base port offset.

All mapped well-known ports are offset by this value.

[default] 7400

[range] [>= 1], but resulting ports must be within the range imposed by the underlying transport.

9.165.2.2 domain_id_gain

```
DDS_Long DDS_RtpsWellKnownPorts_t::domain_id_gain
```

<<***extension***>> *(p. 213)* Tunable domain gain parameter.

Multiplier of the `domain_id`. Together with `participant_id_gain`, it determines the highest `domain_id` and `participant_id` allowed on this network.

In general, there are two ways to setup `domain_id_gain` and `participant_id_gain` parameters.

If `domain_id_gain` > `participant_id_gain`, it results in a port mapping layout where all **DDSDomain↩ Participant** (p. 1619) instances within a single domain occupy a consecutive range of `domain_id_gain` ports. Precisely, all ports occupied by the domain fall within:

```
(port_base + (domain_id_gain * domain_id))
```

and:

```
(port_base + (domain_id_gain * (domain_id + 1)) - 1)
```

Under such a case, the highest `domain_id` is limited only by the underlying transport's maximum port. The highest `participant_id`, however, must satisfy:

```
max_participant_id < (domain_id_gain / participant_id_gain)
```

On the contrary, if `domain_id_gain` <= `participant_id_gain`, it results in a port mapping layout where a given domain's **DDSDomainParticipant** (p. 1619) instances occupy ports spanned across the entire valid port range allowed by the underlying transport. For instance, it results in the following potential mapping:

Mapped Port	Domain Id	Participant ID
higher port number	Domain Id = 1	Participant ID = 2
	Domain Id = 0	Participant ID = 2
	Domain Id = 1	Participant ID = 1
	Domain Id = 0	Participant ID = 1
	Domain Id = 1	Participant ID = 0
lower port number	Domain Id = 0	Participant ID = 0

Under this case, the highest participant_id is limited only by the underlying transport's maximum port. The highest domain_id, however, must satisfy:

```
max_domain_id < (participant_id_gain / domain_id_gain)
```

Additionally, domain_id_gain also determines the range of the port-specific offsets.

```
domain_id_gain > abs(builtin_multicast_port_offset - user_multicast_port_offset)
```

```
domain_id_gain > abs(builtin_unicast_port_offset - user_unicast_port_offset)
```

Violating this may result in port aliasing and undefined discovery behavior.

[default] 250

[range] [> 0], but resulting ports must be within the range imposed by the underlying transport.

9.165.2.3 participant_id_gain

```
DDS_Long DDS_RtpsWellKnownPorts_t::participant_id_gain
```

<<***extension***>> *(p. 213)* Tunable participant gain parameter.

Multiplier of the participant_id. See **DDS_RtpsWellKnownPorts_t::domain_id_gain** (p. 1341) for its implications on the highest domain_id and participant_id allowed on this network.

Additionally, participant_id_gain also determines the range of builtin_unicast_port_offset and user_unicast_port_offset.

```
participant_id_gain > abs(builtin_unicast_port_offset - user_unicast_port_offset)
```

[default] 2

[range] [> 0], but resulting ports must be within the range imposed by the underlying transport.

9.165.2.4 builtin_multicast_port_offset

```
DDS_Long DDS_RtpsWellKnownPorts_t::builtin_multicast_port_offset
```

<<***extension***>> *(p. 213)* Additional offset for **metatraffic** multicast port.

It must be unique from other port-specific offsets.

[default] 0

[range] [>= 0], but resulting ports must be within the range imposed by the underlying transport.

9.165.2.5 builtin_unicast_port_offset

```
DDS_Long DDS_RtpsWellKnownPorts_t::builtin_unicast_port_offset
```

<<***extension***>> *(*p. *213)* Additional offset for **metatraffic** unicast port.

It must be unique from other port-specific offsets.

[default] 10

[range] [>= 0], but resulting ports must be within the range imposed by the underlying transport.

9.165.2.6 user_multicast_port_offset

```
DDS_Long DDS_RtpsWellKnownPorts_t::user_multicast_port_offset
```

<<***extension***>> *(*p. *213)* Additional offset for **usertraffic** multicast port.

It must be unique from other port-specific offsets.

[default] 1

[range] [>= 0], but resulting ports must be within the range imposed by the underlying transport.

9.165.2.7 user_unicast_port_offset

```
DDS_Long DDS_RtpsWellKnownPorts_t::user_unicast_port_offset
```

<<***extension***>> *(*p. *213)* Additional offset for **usertraffic** unicast port.

It must be unique from other port-specific offsets.

[default] 11

[range] [>= 0], but resulting ports must be within the range imposed by the underlying transport.

9.166 DDS_SampleIdentity_t Struct Reference

Type definition for a Sample Identity.

Public Attributes

- struct **DDS_GUID_t writer_guid**
  16-byte identifier identifying the virtual GUID.
- struct **DDS_SequenceNumber_t sequence_number**
  monotonically increasing 64-bit integer that identifies the sample in the data source.

9.166.1 Detailed Description

Type definition for a Sample Identity.

A SampleIdentity defines a pair (Virtual Writer GUID, Sequence Number) that uniquely identifies a sample within a DDS domain and a Topic.

9.167 DDS_SampleInfo Struct Reference

Information that accompanies each sample that is `read` or `taken`.

Public Attributes

- **DDS_SampleStateKind sample_state**

  The sample state of the sample.
- **DDS_ViewStateKind view_state**

  The view state of the instance.
- **DDS_InstanceStateKind instance_state**

  The instance state of the instance.
- struct **DDS_Time_t source_timestamp**

  The timestamp when the sample was written by a DataWriter.
- **DDS_InstanceHandle_t instance_handle**

  Identifies locally the corresponding instance.
- **DDS_InstanceHandle_t publication_handle**

  Identifies locally the DataWriter that modified the instance.
- **DDS_Long disposed_generation_count**

  The disposed generation count of the instance at the time of sample reception.
- **DDS_Long no_writers_generation_count**

  The no writers generation count of the instance at the time of sample reception.
- **DDS_Long sample_rank**

  The sample rank of the sample.
- **DDS_Long generation_rank**

  The generation rank of the sample.
- **DDS_Long absolute_generation_rank**

  The absolute generation rank of the sample.
- **DDS_Boolean valid_data**

  Indicates whether the `DataSample` contains data or else it is only used to communicate a change in the `instance_state` of the instance.
- struct **DDS_Time_t reception_timestamp**

  *<<**extension**>> (p. 432) The timestamp when the sample was committed by a DataReader.*
- struct **DDS_SequenceNumber_t publication_sequence_number**

  *<<**extension**>> (p. 432) The publication sequence number.*
- struct **DDS_SequenceNumber_t reception_sequence_number**

  *<<**extension**>> (p. 432) The reception sequence number when sample was committed by a DataReader*
- struct **DDS_GUID_t original_publication_virtual_guid**

<<***extension***>> (p. 432) *The original publication virtual GUID.*

- struct **DDS_SequenceNumber_t original_publication_virtual_sequence_number**

  <<***extension***>> (p. 432) *The original publication virtual sequence number.*
- struct **DDS_GUID_t related_original_publication_virtual_guid**

  <<***extension***>> (p. 432) *The original publication virtual GUID of a related sample.*
- struct **DDS_SequenceNumber_t related_original_publication_virtual_sequence_number**

  <<***extension***>> (p. 432) *The original publication virtual sequence number of a related sample.*
- **DDS_SampleFlag flag**

  <<***extension***>> (p. 432) *Flags associated with the sample.*
- struct **DDS_GUID_t source_guid**

  <<***extension***>> (p. 432) *The application logical data source associated with the sample.*
- struct **DDS_GUID_t related_source_guid**

  <<***extension***>> (p. 432) *The application logical data source that is related to the sample.*
- struct **DDS_GUID_t related_subscription_guid**

  <<***extension***>> (p. 432) *The related_reader_guid associated with the sample.*
- struct **DDS_GUID_t topic_query_guid**

  <<***extension***>> (p. 432) *The GUID of the* ***DDSTopicQuery*** (p. 1895) *that is related to the sample.*
- struct **DDS_CoherentSetInfo_t** * **coherent_set_info**

  <<***extension***>> (p. 432) *The information about the coherent set that this sample is a part of.*

9.167.1 Detailed Description

Information that accompanies each sample that is `read` or `taken`.

9.167.2 Interpretation of the SampleInfo

The **DDS_SampleInfo** (p. 1344) contains information pertaining to the associated `Data` instance sample including:

- the `sample_state` of the `Data` value (i.e., if it has already been read or not)
- the `view_state` of the related instance (i.e., if the instance is new or not)
- the `instance_state` of the related instance (i.e., if the instance is alive or not)
- **DDS_SampleInfo::valid_data** (p. 1351) flag. This flag indicates whether there is data associated with the sample. Some samples do not contain data indicating only a change on the `instance_state` of the corresponding instance.
- The values of `disposed_generation_count` and `no_writers_generation_count` for the related instance at the time the sample was received. These counters indicate the number of times the instance had become ALIVE (with `instance_state=` **DDS_ALIVE_INSTANCE_STATE** (p. 166)) at the time the sample was received.
- The `sample_rank` and `generation_rank` of the sample within the returned sequence. These ranks provide a preview of the samples that follow within the sequence returned by the `read` or `take` operations.
- The `absolute_generation_rank` of the sample within the **DDSDataReader** (p. 1556). This rank provides a preview of what is available within the **DDSDataReader** (p. 1556).
- The `source_timestamp` of the sample. This is the timestamp provided by the **DDSDataWriter** (p. 1589) at the time the sample was produced.

9.167.3 Interpretation of the SampleInfo disposed_generation_count and no_writers_generation_count

For each instance, RTI Connext internally maintains two counts, the **DDS_SampleInfo::disposed_generation_count** (p. 1349) and **DDS_SampleInfo::no_writers_generation_count** (p. 1349), relative to each DataReader:

- The **DDS_SampleInfo::disposed_generation_count** (p. 1349) and **DDS_SampleInfo::no_writers_generation_count** (p. 1349) are initialized to zero when the **DDSDataReader** (p. 1556) first detects the presence of a never-seen-before instance.
- The **DDS_SampleInfo::disposed_generation_count** (p. 1349) is incremented each time the instance_state of the corresponding instance changes from **DDS_NOT_ALIVE_DISPOSED_INSTANCE_STATE** (p. 166) to **DDS_ALIVE_INSTANCE_STATE** (p. 166).
- The **DDS_SampleInfo::no_writers_generation_count** (p. 1349) is incremented each time the instance_state of the corresponding instance changes from **DDS_NOT_ALIVE_NO_WRITERS_INSTANCE_STATE** (p. 166) to **DDS_ALIVE_INSTANCE_STATE** (p. 166).
- These 'generation counts' are reset to zero when the instance resource is reclaimed.

The **DDS_SampleInfo::disposed_generation_count** (p. 1349) and **DDS_SampleInfo::no_writers_generation_count** (p. 1349) available in the **DDS_SampleInfo** (p. 1344) capture a snapshot of the corresponding counters at the time the sample was received.

9.167.4 Interpretation of the SampleInfo sample_rank, generation_rank and absolute_generation_rank

The **DDS_SampleInfo::sample_rank** (p. 1350) and **DDS_SampleInfo::generation_rank** (p. 1350) available in the **DDS_SampleInfo** (p. 1344) are computed based solely on the actual samples in the ordered collection returned by read or take.

- The **DDS_SampleInfo::sample_rank** (p. 1350) indicates the number of samples of the same instance that follow the current one in the collection.
- The **DDS_SampleInfo::generation_rank** (p. 1350) available in the **DDS_SampleInfo** (p. 1344) indicates the difference in "generations" between the sample (S) and the Most Recent Sample of the same instance that appears in the returned Collection (MRSIC). That is, it counts the number of times the instance transitioned from not-alive to alive in the time from the reception of the S to the reception of MRSIC.
- These 'generation ranks' are reset to zero when the instance resource is reclaimed.

The **DDS_SampleInfo::generation_rank** (p. 1350) is computed using the formula:

```
generation_rank = (MRSIC.disposed_generation_count

                       + MRSIC.no_writers_generation_count)

                  - (S.disposed_generation_count

                       + S.no_writers_generation_count)
```

The **DDS_SampleInfo::absolute_generation_rank** (p. 1350) available in the **DDS_SampleInfo** (p. 1344) indicates the difference in "generations" between the sample (S) and the Most Recent Sample of the same instance that the middleware has received (MRS). That is, it counts the number of times the instance transitioned from not-alive to alive in the time from the reception of the S to the time when the read or take was called.

```
absolute_generation_rank = (MRS.disposed_generation_count

                                + MRS.no_writers_generation_count)

                           - (S.disposed_generation_count

                                + S.no_writers_generation_count)
```

9.167.5 Interpretation of the SampleInfo counters and ranks

These counters and ranks allow the application to distinguish samples belonging to different "generations" of the instance. Note that it is possible for an instance to transition from not-alive to alive (and back) several times before the application accesses the data by means of read or take. In this case, the returned collection may contain samples that cross generations (i.e. some samples were received before the instance became not-alive, other after the instance re-appeared again). Using the information in the **DDS_SampleInfo** (p. 1344), the application can anticipate what other information regarding the same instance appears in the returned collection, as well as in the infrastructure and thus make appropriate decisions.

For example, an application desiring to only consider the most current sample for each instance would only look at samples with sample_rank == 0. Similarly, an application desiring to only consider samples that correspond to the latest generation in the collection will only look at samples with generation_rank == 0. An application desiring only samples pertaining to the latest generation available will ignore samples for which absolute_generation_rank != 0. Other application-defined criteria may also be used.

See also

DDS_SampleStateKind (p. 161), **DDS_InstanceStateKind** (p. 165), **DDS_ViewStateKind** (p. 163), **DDS_↩ SampleInfo::valid_data** (p. 1351)

"Statechart of the \p instance_state and \p view_state of a single instance"

9.167.6 Member Data Documentation

9.167.6.1 sample_state

```
DDS_SampleStateKind DDS_SampleInfo::sample_state
```

The sample state of the sample.

Indicates whether or not the corresponding data sample has already been read.

See also

DDS_SampleStateKind (p. 161)

9.167.6.2 view_state

```
DDS_ViewStateKind DDS_SampleInfo::view_state
```

The view state of the instance.

Indicates whether the **DDSDataReader** (p. 1556) has already seen samples for the most-current generation of the related instance.

See also

DDS_ViewStateKind (p. 163)

9.167.6.3 instance_state

```
DDS_InstanceStateKind DDS_SampleInfo::instance_state
```

The instance state of the instance.

Indicates whether the instance is currently in existence or, if it has been disposed, the reason why it was disposed.

See also

DDS_InstanceStateKind (p. 165)

9.167.6.4 source_timestamp

```
struct DDS_Time_t DDS_SampleInfo::source_timestamp
```

The timestamp when the sample was written by a DataWriter.

9.167.6.5 instance_handle

```
DDS_InstanceHandle_t DDS_SampleInfo::instance_handle
```

Identifies locally the corresponding instance.

The handle is equal to **DDS_HANDLE_NIL** (p. 78) for unkeyed topics.

9.167.6.6 publication_handle

```
DDS_InstanceHandle_t DDS_SampleInfo::publication_handle
```

Identifies locally the DataWriter that modified the instance.

The `publication_handle` is the same **DDS_InstanceHandle_t** (p. 76) that is returned by the operation **DDSDataReader::get_matched_publications** (p. 1567) and can also be used as a parameter to the operation **DDSDataReader::get_matched_publication_data** (p. 1568).

9.167.6.7 disposed_generation_count

```
DDS_Long DDS_SampleInfo::disposed_generation_count
```

The disposed generation count of the instance at the time of sample reception.

Indicates how many times the `instance_state` of the corresponding instance changed from **DDS_NOT_ALIVE↩ _DISPOSED_INSTANCE_STATE** (p. 166) to **DDS_ALIVE_INSTANCE_STATE** (p. 166). The counter is reset when the instance resource is reclaimed (removed from the DataReader cache).

See also

Interpretation of the SampleInfo disposed_generation_count and no_writers_generation_count (p. 1346)
Interpretation of the SampleInfo counters and ranks (p. 1347)

9.167.6.8 no_writers_generation_count

DDS_Long DDS_SampleInfo::no_writers_generation_count

The no writers generation count of the instance at the time of sample reception.

Indicates how many times the `instance_state` of the corresponding instance changed from **DDS_NOT_ALIVE↩ _NO_WRITERS_INSTANCE_STATE** (p. 166) to **DDS_ALIVE_INSTANCE_STATE** (p. 166). The counter is reset when the instance resource is reclaimed (removed from the DataReader cache).

See also

Interpretation of the SampleInfo disposed_generation_count and no_writers_generation_count (p. 1346)
Interpretation of the SampleInfo counters and ranks (p. 1347)

9.167.6.9 sample_rank

DDS_Long DDS_SampleInfo::sample_rank

The sample rank of the sample.

Indicates the number of samples related to the same instance that follow in the collection returned by `read` or `take`.

See also

Interpretation of the SampleInfo sample_rank, generation_rank and absolute_generation_rank (p. 1346)
Interpretation of the SampleInfo counters and ranks (p. 1347)

9.167.6.10 generation_rank

DDS_Long DDS_SampleInfo::generation_rank

The generation rank of the sample.

Indicates the generation difference (number of times the instance was NOT_ALIVE and become alive again) between the time the sample was received and the time the most recent sample in the collection related to the same instance was received.

See also

Interpretation of the SampleInfo sample_rank, generation_rank and absolute_generation_rank (p. 1346)
Interpretation of the SampleInfo counters and ranks (p. 1347)

9.167.6.11 absolute_generation_rank

DDS_Long DDS_SampleInfo::absolute_generation_rank

The absolute generation rank of the sample.

Indicates the generation difference (number of times the instance was disposed and become alive again) between the time the sample was received, and the time the most recent sample (which may not be in the returned collection) related to the same instance was received.

See also

Interpretation of the SampleInfo sample_rank, generation_rank and absolute_generation_rank (p. 1346)
Interpretation of the SampleInfo counters and ranks (p. 1347)

9.167.6.12 valid_data

DDS_Boolean DDS_SampleInfo::valid_data

Indicates whether the DataSample contains data or else it is only used to communicate a change in the instance_state of the instance.

Normally each DataSample contains both a **DDS_SampleInfo** (p. 1344) and some Data. However there are situations where a DataSample contains only the **DDS_SampleInfo** (p. 1344) and does not have any associated data. This occurs when the RTI Connext notifies the application of a change of state for an instance that was caused by some internal mechanism (such as a timeout) for which there is no associated data. An example of this situation is when the RTI Connext detects that an instance has no writers and changes the corresponding instance_state to **DDS_NOT_ALIVE_NO_WRITERS_INSTANCE_STATE** (p. 166).

The application can distinguish whether a particular DataSample has data by examining the value of the **DDS_SampleInfo::valid_data** (p. 1351). If this flag is set to **DDS_BOOLEAN_TRUE** (p. 513), then the DataSample contains valid Data. If the flag is set to **DDS_BOOLEAN_FALSE** (p. 513), the DataSample contains no Data.

To ensure correctness and portability, the valid_data flag must be examined by the application prior to accessing the Data associated with the DataSample and if the flag is set to **DDS_BOOLEAN_FALSE** (p. 513), the application should not access the Data associated with the DataSample, that is, the application should access only the **DDS_SampleInfo** (p. 1344).

9.167.6.13 reception_timestamp

struct **DDS_Time_t** DDS_SampleInfo::reception_timestamp

<<***extension***>> *(p. 432)* The timestamp when the sample was committed by a DataReader.

9.167.6.14 publication_sequence_number

```
struct DDS_SequenceNumber_t DDS_SampleInfo::publication_sequence_number
```

<<***extension***>> *(p. 432)* The publication sequence number.

9.167.6.15 reception_sequence_number

```
struct DDS_SequenceNumber_t DDS_SampleInfo::reception_sequence_number
```

<<***extension***>> *(p. 432)* The reception sequence number when sample was committed by a DataReader

9.167.6.16 original_publication_virtual_guid

```
struct DDS_GUID_t DDS_SampleInfo::original_publication_virtual_guid
```

<<***extension***>> *(p. 432)* The original publication virtual GUID.

If the **DDS_PresentationQosPolicy::access_scope** (p. 1252) of the **DDSPublisher** (p. 1818) is **DDS_GROUP_↩ PRESENTATION_QOS** (p. 616), this field contains the **DDSPublisher** (p. 1818) virtual GUID that uniquely identifies the DataWriter group.

See also

connext::Sample::identity() (p. 2185)

9.167.6.17 original_publication_virtual_sequence_number

```
struct DDS_SequenceNumber_t DDS_SampleInfo::original_publication_virtual_sequence_number
```

<<***extension***>> *(p. 432)* The original publication virtual sequence number.

If the **DDS_PresentationQosPolicy::access_scope** (p. 1252) of the **DDSPublisher** (p. 1818) is **DDS_GROUP_↩ PRESENTATION_QOS** (p. 616), this field contains the **DDSPublisher** (p. 1818) virtual sequence number that uniquely identifies a sample within the DataWriter group.

See also

connext::Sample::identity() (p. 2185)

9.167.6.18 related_original_publication_virtual_guid

```
struct DDS_GUID_t DDS_SampleInfo::related_original_publication_virtual_guid
```

<<***extension***>> *(p. 432)* The original publication virtual GUID of a related sample.

See also

connext::Sample::related_identity() (p. 2185)

9.167.6.19 related_original_publication_virtual_sequence_number

```
struct DDS_SequenceNumber_t DDS_SampleInfo::related_original_publication_virtual_sequence_number
```

<<***extension***>> *(p. 432)* The original publication virtual sequence number of a related sample.

See also

connext::Sample::related_identity() (p. 2185)

9.167.6.20 flag

```
DDS_SampleFlag DDS_SampleInfo::flag
```

<<***extension***>> *(p. 432)* Flags associated with the sample.

The flags can be set by using the field **DDS_WriteParams_t::flag** (p. 1522) when writing a sample using the method **FooDataWriter::write_w_params** (p. 1956).

9.167.6.21 source_guid

```
struct DDS_GUID_t DDS_SampleInfo::source_guid
```

<<***extension***>> *(p. 432)* The application logical data source associated with the sample.

The source_guid can be set by using the field **DDS_WriteParams_t::source_guid** (p. 1522) when writing a sample using the method **FooDataWriter::write_w_params** (p. 1956).

See also

DDS_WriteParams_t::source_guid (p. 1522) for additional information on **source_guid** (p. 1353) and when it is used.

9.167.6.22 related_source_guid

```
struct DDS_GUID_t DDS_SampleInfo::related_source_guid
```

<<***extension***>> *(p. 432)* The application logical data source that is related to the sample.

The related_source_guid can be set by using the field **DDS_WriteParams_t::related_source_guid** (p. 1523) when writing a sample using the method **FooDataWriter::write_w_params** (p. 1956).

9.167.6.23 related_subscription_guid

```
struct DDS_GUID_t DDS_SampleInfo::related_subscription_guid
```

<<***extension***>> *(p. 432)* The related_reader_guid associated with the sample.

The related_reader_guid can be set by using the field **DDS_WriteParams_t::related_reader_guid** (p. 1523) when writing a sample using the method **FooDataWriter::write_w_params** (p. 1956).

9.167.6.24 topic_query_guid

```
struct DDS_GUID_t DDS_SampleInfo::topic_query_guid
```

<<***extension***>> *(p. 432)* The GUID of the **DDSTopicQuery** (p. 1895) that is related to the sample.

This GUID indicates whether a sample is part of the response to a **DDSTopicQuery** (p. 1895) or a regular ("live") sample:

- If the sample was written for the TopicQuery stream, this field contains the GUID of the target TopicQuery.
- If the sample was written for the live stream, this field will be set to **DDS_GUID_UNKNOWN** (p. 527).

9.167.6.25 coherent_set_info

```
struct DDS_CoherentSetInfo_t* DDS_SampleInfo::coherent_set_info
```

<<***extension***>> *(p. 432)* The information about the coherent set that this sample is a part of.

This field is set for all samples that are part of a coherent set. Coherent sets are initiated using the operation **DDSPublisher::begin_coherent_changes** (p. 1832) and finalized using the operation **DDSPublisher::end_↩ coherent_changes** (p. 1833).

See also

DDSPublisher::begin_coherent_changes (p. 1832) for additional information on coherent sets.

9.168 DDS_SampleInfoSeq Struct Reference

Declares IDL `sequence` < **DDS_SampleInfo** (p. 1344) > .

9.168.1 Detailed Description

Declares IDL `sequence` < **DDS_SampleInfo** (p. 1344) > .

See also

FooSeq (p. 1965)

Examples

HelloWorld_subscriber.cxx.

9.169 DDS_SampleLostStatus Struct Reference

DDS_SAMPLE_LOST_STATUS (p. 541)

Public Attributes

- **DDS_Long total_count**

  *Total cumulative count of all samples lost across all instances of data published under the **DDSTopic** (p. 1885).*
- **DDS_Long total_count_change**

  The incremental number of samples lost since the last time the listener was called or the status was read.
- **DDS_SampleLostStatusKind last_reason**

  *<<**extension**>> (p. 432) Reason why the last sample was lost.*

9.169.1 Detailed Description

DDS_SAMPLE_LOST_STATUS (p. 541)

9.169.2 Member Data Documentation

9.169.2.1 total_count

```
DDS_Long DDS_SampleLostStatus::total_count
```

Total cumulative count of all samples lost across all instances of data published under the **DDSTopic** (p. 1885).

9.169.2.2 total_count_change

```
DDS_Long DDS_SampleLostStatus::total_count_change
```

The incremental number of samples lost since the last time the listener was called or the status was read.

9.169.2.3 last_reason

```
DDS_SampleLostStatusKind DDS_SampleLostStatus::last_reason
```

<<***extension***>> *(*p. *432)* Reason why the last sample was lost.

See also

DDS_SampleLostStatusKind (p. 140)

9.170 DDS_SampleRejectedStatus Struct Reference

DDS_SAMPLE_REJECTED_STATUS (p. 541)

Public Attributes

- **DDS_Long total_count**
  Total cumulative count of samples rejected by the ***DDSDataReader*** *(*p. *1556).*
- **DDS_Long total_count_change**
  The incremental number of samples rejected since the last time the listener was called or the status was read.
- **DDS_SampleRejectedStatusKind last_reason**
  Reason for rejecting the last sample rejected.
- **DDS_InstanceHandle_t last_instance_handle**
  Handle to the instance being updated by the last sample that was rejected.

9.170.1 Detailed Description

DDS_SAMPLE_REJECTED_STATUS (p. 541)

9.170.2 Member Data Documentation

9.170.2.1 total_count

DDS_Long DDS_SampleRejectedStatus::total_count

Total cumulative count of samples rejected by the **DDSDataReader** (p. 1556).

9.170.2.2 total_count_change

DDS_Long DDS_SampleRejectedStatus::total_count_change

The incremental number of samples rejected since the last time the listener was called or the status was read.

9.170.2.3 last_reason

DDS_SampleRejectedStatusKind DDS_SampleRejectedStatus::last_reason

Reason for rejecting the last sample rejected.

See also

DDS_SampleRejectedStatusKind (p. 144)

9.170.2.4 last_instance_handle

DDS_InstanceHandle_t DDS_SampleRejectedStatus::last_instance_handle

Handle to the instance being updated by the last sample that was rejected.

If the sample was rejected because of **DDS_REJECTED_BY_DECODE_FAILURE** (p. 146) and the **DDSData↩ Writer** (p. 1589) set **DDS_DataWriterProtocolQosPolicy::disable_inline_keyhash** (p. 895) to **DDS_BOOLEAN_↩ TRUE** (p. 513), then the last_instance_handle may not be correct if the sample was encrypted.

9.171 DDS_SequenceNumber_t Struct Reference

Type for *sequence* number representation.

Public Attributes

- **DDS_Long high**

  The most significant part of the sequence number.
- **DDS_UnsignedLong low**

  The least significant part of the sequence number.

9.171.1 Detailed Description

Type for *sequence* number representation.

Represents a 64-bit sequence number.

9.171.2 Member Data Documentation

9.171.2.1 high

DDS_Long DDS_SequenceNumber_t::high

The most significant part of the sequence number.

9.171.2.2 low

DDS_UnsignedLong DDS_SequenceNumber_t::low

The least significant part of the sequence number.

9.172 DDS_ServiceQosPolicy Struct Reference

<<***extension***>> *(*p. *213)* Service or tool associated with a DDS entity.

Public Attributes

- **DDS_ServiceQosPolicyKind kind**

  <<***extension***>> *(*p. *213) The kind of service.*

9.172.1 Detailed Description

<<***extension***>> *(p. 213)* Service or tool associated with a DDS entity.

This QoS policy is intended to be used by RTI infrastructure services and RTI tools.

User applications should not modify its value.

Entity:

DDSDomainParticipant (p. 1619), **DDSDataReader** (p. 1556), **DDSDataWriter** (p. 1589)

Properties:

RxO (p. 556) = NO
Changeable (p. 556) = **UNTIL ENABLE** (p. 556)

9.172.2 Member Data Documentation

9.172.2.1 kind

DDS_ServiceQosPolicyKind DDS_ServiceQosPolicy::kind

<<***extension***>> *(p. 213)* The kind of service.

[default] DDS_NO_SERVICE_QOS (p. 637)

Categories

Immutable (p. 333), **Discovery Metadata** (p. 377), **Debuggability** (p. 266)

9.173 DDS_ServiceRequest Struct Reference

A request coming from one of the built-in services.

Public Attributes

- **DDS_Long service_id**
  The id of the service that the request was sent on.
- struct **DDS_GUID_t instance_id**
  Each ServiceRequest is keyed on the instance_id.
- struct **DDS_OctetSeq request_body**
  Service-specific information.

9.173.1 Detailed Description

A request coming from one of the built-in services.

Data associated with the built-in topic **DDS_SERVICE_REQUEST_TOPIC_NAME** (p. 499). It contains service-specific information.

See also

DDS_SERVICE_REQUEST_TOPIC_NAME (p. 499)

DDSParticipantBuiltinTopicDataDataReader (p. 1816)

9.173.2 Member Data Documentation

9.173.2.1 service_id

DDS_Long DDS_ServiceRequest::service_id

The id of the service that the request was sent on.

There can be multiple services that use the built-in ServiceRequest topic. The service_id identifies which service a specific request was sent from.

See also

DDS_UNKNOWN_SERVICE_REQUEST_ID (p. 498)

DDS_TOPIC_QUERY_SERVICE_REQUEST_ID (p. 498)

9.173.2.2 instance_id

struct **DDS_GUID_t** DDS_ServiceRequest::instance_id

Each ServiceRequest is keyed on the instance_id.

The instance_id provides a way for users to differentiate between different requests coming from the same service.

9.173.2.3 request_body

```
struct DDS_OctetSeq DDS_ServiceRequest::request_body
```

Service-specific information.

Each service uses the request_body field to send information specific to that service in the form of an opaque sequence of bytes. Each service provides a helper function that will deserialize the information from the request body.

See also

DDSTopicQueryHelper::topic_query_data_from_service_request (p. 1896)

9.174 DDS_ServiceRequestAcceptedStatus Struct Reference

DDS_SERVICE_REQUEST_ACCEPTED_STATUS (p. 544)

Public Attributes

- **DDS_Long total_count**

  *The total cumulative number of ServiceRequests that have been accepted by a **DDSDataWriter** (p. 1589).*
- **DDS_Long total_count_change**

  The incremental changes in total_count since the last time the listener was called or the status was read.
- **DDS_Long current_count**

  *The current number of ServiceRequests that have been accepted by this **DDSDataWriter** (p. 1589).*
- **DDS_Long current_count_change**

  The change in current_count since the last time the listener was called or the status was read.
- **DDS_InstanceHandle_t last_request_handle**

  *A handle to the last **DDS_ServiceRequest** (p. 1359) that caused the **DDSDataWriter** (p. 1589)'s status to change.*
- **DDS_Long service_id**

  ID of the service to which the accepted Request belongs.

9.174.1 Detailed Description

DDS_SERVICE_REQUEST_ACCEPTED_STATUS (p. 544)

Currently, the only service that causes the ServiceRequestAcceptedStatus to be triggered is the **DDSTopicQuery** (p. 1895) service. A **DDS_ServiceRequest** (p. 1359) is accepted when a **DDSDataWriter** (p. 1589) matches with a **DDSDataReader** (p. 1556) that has created a **DDSTopicQuery** (p. 1895).

This status is also changed (and the listener, if any, called) when a ServiceRequest has been cancelled, or deleted. This will happen when a **DDSDataReader** (p. 1556) deletes a TopicQuery using **DDSDataReader::delete_topic_↩query** (p. 1578).

9.174.2 Member Data Documentation

9.174.2.1 total_count

DDS_Long DDS_ServiceRequestAcceptedStatus::total_count

The total cumulative number of ServiceRequests that have been accepted by a **DDSDataWriter** (p. 1589).

This number increases whenever a new request is accepted. It does not change when a request is cancelled.

9.174.2.2 total_count_change

DDS_Long DDS_ServiceRequestAcceptedStatus::total_count_change

The incremental changes in total_count since the last time the listener was called or the status was read.

9.174.2.3 current_count

DDS_Long DDS_ServiceRequestAcceptedStatus::current_count

The current number of ServiceRequests that have been accepted by this **DDSDataWriter** (p. 1589).

This number increases when a new request is accepted and decreases when an existing request is cancelled.

9.174.2.4 current_count_change

DDS_Long DDS_ServiceRequestAcceptedStatus::current_count_change

The change in current_count since the last time the listener was called or the status was read.

9.174.2.5 last_request_handle

DDS_InstanceHandle_t DDS_ServiceRequestAcceptedStatus::last_request_handle

A handle to the last **DDS_ServiceRequest** (p. 1359) that caused the **DDSDataWriter** (p. 1589)'s status to change.

9.174.2.6 service_id

```
DDS_Long DDS_ServiceRequestAcceptedStatus::service_id
```

ID of the service to which the accepted Request belongs.

See also

DDS_TOPIC_QUERY_SERVICE_REQUEST_ID (p. 498)

9.175 DDS_ServiceRequestSeq Struct Reference

Instantiates **FooSeq** (p. 1965) < **DDS_ServiceRequest** (p. 1359) > .

9.175.1 Detailed Description

Instantiates **FooSeq** (p. 1965) < **DDS_ServiceRequest** (p. 1359) > .

Instantiates:

<<***generic***>> *(*p. *433)* **FooSeq** (p. 1965)

See also

DDS_ServiceRequest (p. 1359)

9.176 DDS_ShortSeq Struct Reference

Instantiates **FooSeq** (p. 1965) < **DDS_Short** (p. 514) >

9.176.1 Detailed Description

Instantiates **FooSeq** (p. 1965) < **DDS_Short** (p. 514) >

Instantiates:

<<***generic***>> *(*p. *433)* **FooSeq** (p. 1965)

See also

DDS_Short (p. 514)

FooSeq (p. 1965)

9.177 DDS_StringSeq Struct Reference

Instantiates **FooSeq** (p. 1965) < char* > with value type semantics.

9.177.1 Detailed Description

Instantiates **FooSeq** (p. 1965) < char* > with value type semantics.

StringSeq is a sequence that contains strings.

Even though the element type is a char*, i.e. a pointer, the sequence semantically behaves as a sequence of char* *value* types. When a **DDS_StringSeq** (p. 1364) is copied or deleted, the contained strings are also respectively copied or deleted.

Important: Users of this type must understand its memory management contract.

- Ownership of this sequence's buffer implies ownership of the pointers stored in that buffer; a loan of the buffer implies lack of ownership of the pointers. In other words, for a type **FooSeq** (p. 1965) where **Foo** (p. 1916) is a pointer, ownership of **Foo** (p. 1916) implies ownership of * **Foo** (p. 1916). In other words, deleting a string sequence that owns its memory implies the deletion of all strings in that sequence. See **FooSeq::loan_contiguous** (p. 1976) for more information about sequence memory ownership.
- The second important rule is that non-NULL strings are *assumed to be of sufficient size* to store the necessary characters. This is a dangerous rule, but it cannot be avoided because a string doesn't store the amount of memory it has. The only other alternative is to always free and re-allocate memory. Not only would this latter policy be very expensive, but it would essentially render any loaned **DDS_StringSeq** (p. 1364) immutable, since to modify any string in it would require freeing and re-allocating that string, which would violate the first principle discussed above.

It is also worth noting that the element type of a string sequence is char*, not const char*. It is therefore incorrect and dangerous, for example, to insert a string literal into a string sequence without first copying it into mutable memory.

In order to guarantee correct behavior, it is recommended that the contained elements always be manipulated using the string support API's described in **String Support** (p. 757).

See also

String Support (p. 757)

Instantiates:

<<***generic***>> *(p. 433)* **FooSeq** (p. 1965)

See also

FooSeq (p. 1965)

9.178 DDS_StructMember Struct Reference

A description of a member of a struct.

Public Attributes

- char * **name**

  The name of the struct member.
- const **DDS_TypeCode** * **type**

  The type of the struct member.
- **DDS_Boolean is_pointer**

  Indicates whether the struct member is a pointer or not.
- **DDS_Short bits**

  Number of bits of a bitfield member.
- **DDS_Boolean is_key**

  Indicates if the struct member is a key member or not.
- **DDS_Long id**

  The member ID.
- **DDS_Boolean is_optional**

  Indicates if the struct member is optional or required.
- **DDS_Boolean is_must_understand**

  Indicates if the struct member is annotated as "must understand".

9.178.1 Detailed Description

A description of a member of a struct.

See also

DDS_StructMemberSeq (p. 1367)

DDS_TypeCodeFactory::create_struct_tc (p. 1480)

9.178.2 Member Data Documentation

9.178.2.1 name

```
char* DDS_StructMember::name
```

The name of the struct member.

Cannot be NULL.

9.178.2.2 type

```
const  DDS_TypeCode* DDS_StructMember::type
```

The type of the struct member.

Cannot be NULL.

9.178.2.3 is_pointer

```
DDS_Boolean DDS_StructMember::is_pointer
```

Indicates whether the struct member is a pointer or not.

9.178.2.4 bits

```
DDS_Short DDS_StructMember::bits
```

Number of bits of a bitfield member.

If the struct member is a bitfield, this field contains the number of bits of the bitfield. Otherwise, bits should contain **DDS_TYPECODE_NOT_BITFIELD** (p. 83).

9.178.2.5 is_key

```
DDS_Boolean DDS_StructMember::is_key
```

Indicates if the struct member is a key member or not.

9.178.2.6 id

```
DDS_Long DDS_StructMember::id
```

The member ID.

Use **DDS_TYPECODE_MEMBER_ID_INVALID** (p. 83) to have the member ID automatically assigned.

9.178.2.7 is_optional

```
DDS_Boolean DDS_StructMember::is_optional
```

Indicates if the struct member is optional or required.

9.178.2.8 is_must_understand

```
DDS_Boolean DDS_StructMember::is_must_understand
```

Indicates if the struct member is annotated as "must understand".

9.179 DDS_StructMemberSeq Struct Reference

Defines a sequence of struct members.

9.179.1 Detailed Description

Defines a sequence of struct members.

See also

DDS_StructMember (p. 1365)

FooSeq (p. 1965)

DDS_TypeCodeFactory::create_struct_tc (p. 1480)

9.180 DDS_SubscriberQos Struct Reference

QoS policies supported by a **DDSSubscriber** (p. 1860) entity.

Public Member Functions

- bool **operator==** (const **DDS_SubscriberQos** &r) const
  Compares two SubscriberQos objects for equality.
- bool **operator!=** (const **DDS_SubscriberQos** &r) const
  Compares two SubscriberQos objects for inequality.
- **DDS_ReturnCode_t print** () const
  *Prints this **DDS_SubscriberQos** (p. 1367) to stdout.*
- **DDS_ReturnCode_t to_string** (char *string, **DDS_UnsignedLong** &string_size) const
  *Obtains a string representation of this **DDS_SubscriberQos** (p. 1367).*
- **DDS_ReturnCode_t to_string** (char *string, **DDS_UnsignedLong** &string_size, const **DDS_SubscriberQos** &base) const
  *Obtains a string representation of this **DDS_SubscriberQos** (p. 1367).*
- **DDS_ReturnCode_t to_string** (char *string, **DDS_UnsignedLong** &string_size, const **DDS_QosPrintFormat** &format) const
  *Obtains a string representation of this **DDS_SubscriberQos** (p. 1367).*
- **DDS_ReturnCode_t to_string** (char *string, **DDS_UnsignedLong** &string_size, const **DDS_SubscriberQos** &base, const **DDS_QosPrintFormat** &format) const
  *Obtains a string representation of this **DDS_SubscriberQos** (p. 1367).*
- **DDS_ReturnCode_t to_string** (char *string, **DDS_UnsignedLong** &string_size, const **DDS_QosPrintAll_t** &) const
  *Obtains a string representation of this **DDS_SubscriberQos** (p. 1367).*
- **DDS_ReturnCode_t to_string** (char *string, **DDS_UnsignedLong** &string_size, const **DDS_QosPrintAll_t** &, const **DDS_QosPrintFormat** &format) const
  *Obtains a string representation of this **DDS_SubscriberQos** (p. 1367).*

Public Attributes

- struct **DDS_PresentationQosPolicy presentation**
  *Presentation policy, **PRESENTATION** (p. 615).*
- struct **DDS_PartitionQosPolicy partition**
  *<<**basic**>> (p. 201) Partition policy, **PARTITION** (p. 614).*
- struct **DDS_GroupDataQosPolicy group_data**
  *Group data policy, **GROUP_DATA** (p. 604).*
- struct **DDS_EntityFactoryQosPolicy entity_factory**
  *<<**basic**>> (p. 201) Entity factory policy, **ENTITY_FACTORY** (p. 600).*
- struct **DDS_EntityNameQosPolicy subscriber_name**
  *<<**extension**>> (p. 432) EntityName policy, **ENTITY_NAME** (p. 601).*

9.180.1 Detailed Description

QoS policies supported by a **DDSSubscriber** (p. 1860) entity.

You must set certain members in a consistent manner:

length of **DDS_GroupDataQosPolicy::value** (p. 1161) <= **DDS_DomainParticipantResourceLimitsQosPolicy::subscriber_group_data_max_length** (p. 992)

length of **DDS_PartitionQosPolicy::name** (p. 1242) <= **DDS_DomainParticipantResourceLimitsQosPolicy::max_partitions** (p. 993)

combined number of characters (including terminating 0) in **DDS_PartitionQosPolicy::name** (p. 1242) <= **DDS_DomainParticipantResourceLimitsQosPolicy::max_partition_cumulative_characters** (p. 993)

If any of the above are not true, **DDSSubscriber::set_qos** (p. 1877) and **DDSSubscriber::set_qos_with_profile** (p. 1878) will fail with **DDS_RETCODE_INCONSISTENT_POLICY** (p. 533)

9.180.2 Member Function Documentation

9.180.2.1 operator==()

```
bool DDS_SubscriberQos::operator== (
            const  DDS_SubscriberQos & r ) const  [inline]
```

Compares two SubscriberQos objects for equality.

See also

DDS_SubscriberQos_equals (p. 131)

References **DDS_SubscriberQos_equals()**.

9.180.2.2 operator"!=()

```
bool DDS_SubscriberQos::operator!= (
            const  DDS_SubscriberQos & r ) const  [inline]
```

Compares two SubscriberQos objects for inequality.

See also

DDS_SubscriberQos_equals (p. 131)

References **DDS_SubscriberQos_equals()**.

9.180.3 Member Data Documentation

9.180.3.1 presentation

```
struct  DDS_PresentationQosPolicy DDS_SubscriberQos::presentation
```

Presentation policy, **PRESENTATION** (p. 615).

9.180.3.2 partition

```
struct  DDS_PartitionQosPolicy DDS_SubscriberQos::partition
```

<<***basic***>> *(p. 201)* Partition policy, **PARTITION** (p. 614).

9.180.3.3 group_data

```
struct  DDS_GroupDataQosPolicy DDS_SubscriberQos::group_data
```

Group data policy, **GROUP_DATA** (p. 604).

9.180.3.4 entity_factory

`struct` **DDS_EntityFactoryQosPolicy** `DDS_SubscriberQos::entity_factory`

<<***basic***>> *(p. 201)* Entity factory policy, **ENTITY_FACTORY** (p. 600).

9.180.3.5 subscriber_name

`struct` **DDS_EntityNameQosPolicy** `DDS_SubscriberQos::subscriber_name`

<<***extension***>> *(p. 432)* EntityName policy, **ENTITY_NAME** (p. 601).

9.181 DDS_SubscriptionBuiltinTopicData Struct Reference

Entry created when a **DDSDataReader** (p. 1556) is discovered in association with its Subscriber.

Public Attributes

- **DDS_BuiltinTopicKey_t key**
  DCPS key to distinguish entries.
- **DDS_BuiltinTopicKey_t participant_key**
  DCPS key of the participant to which the DataReader belongs.
- char * **topic_name**
  Name of the related ***DDSTopic*** *(p. 1885).*
- char * **type_name**
  Name of the type attached to the ***DDSTopic*** *(p. 1885).*
- struct **DDS_DurabilityQosPolicy durability**
  Policy of the corresponding DataReader.
- struct **DDS_DeadlineQosPolicy deadline**
  Policy of the corresponding DataReader.
- struct **DDS_LatencyBudgetQosPolicy latency_budget**
  Policy of the corresponding DataReader.
- struct **DDS_LivelinessQosPolicy liveliness**
  Policy of the corresponding DataReader.
- struct **DDS_ReliabilityQosPolicy reliability**
  Policy of the corresponding DataReader.
- struct **DDS_OwnershipQosPolicy ownership**
  Policy of the corresponding DataReader.
- struct **DDS_DestinationOrderQosPolicy destination_order**
  Policy of the corresponding DataReader.
- struct **DDS_UserDataQosPolicy user_data**
  Policy of the corresponding DataReader.

- struct **DDS_TimeBasedFilterQosPolicy time_based_filter**

  Policy of the corresponding DataReader.
- struct **DDS_PresentationQosPolicy presentation**

  Policy of the Subscriber to which the DataReader belongs.
- struct **DDS_PartitionQosPolicy partition**

  Policy of the Subscriber to which the DataReader belongs.
- struct **DDS_TopicDataQosPolicy topic_data**

  Policy of the related Topic.
- struct **DDS_GroupDataQosPolicy group_data**

  Policy of the Subscriber to which the DataReader belongs.
- struct **DDS_TypeConsistencyEnforcementQosPolicy type_consistency**

  Policy of the corresponding DataReader.
- struct **DDS_DataRepresentationQosPolicy representation**

  Data representation policy of the corresponding DataReader.
- **DDS_DataTagQosPolicy data_tags**

  Tags of the corresponding DataReader.
- struct **DDS_TypeCode** * **type_code**

  *<<**extension**>> (p. 432) Type code information of the corresponding Topic.*
- **DDS_BuiltinTopicKey_t subscriber_key**

  *<<**extension**>> (p. 432) DCPS key of the subscriber to which the DataReader belongs.*
- struct **DDS_PropertyQosPolicy property**

  *<<**extension**>> (p. 432) Properties of the corresponding DataReader.*
- struct **DDS_LocatorSeq unicast_locators**

  *<<**extension**>> (p. 432) Custom unicast locators that the endpoint can specify. The default locators will be used if this is not specified.*
- struct **DDS_LocatorSeq multicast_locators**

  *<<**extension**>> (p. 432) Custom multicast locators that the endpoint can specify. The default locators will be used if this is not specified.*
- struct **DDS_ContentFilterProperty_t content_filter_property**

  *<<**extension**>> (p. 432) This field provides all the required information to enable content filtering on the Writer side.*
- struct **DDS_GUID_t virtual_guid**

  *<<**extension**>> (p. 432) Virtual GUID associated to the DataReader.*
- struct **DDS_ServiceQosPolicy service**

  *<<**extension**>> (p. 432) Policy of the corresponding DataReader.*
- **DDS_ProtocolVersion_t rtps_protocol_version**

  *<<**extension**>> (p. 432) Version number of the RTPS wire protocol used.*
- struct **DDS_VendorId_t rtps_vendor_id**

  *<<**extension**>> (p. 432) ID of vendor implementing the RTPS wire protocol.*
- struct **DDS_ProductVersion_t product_version**

  *<<**extension**>> (p. 432) This is a vendor specific parameter. It gives the current version of RTI Connext*
- **DDS_Boolean disable_positive_acks**

  *<<**extension**>> (p. 432) This is a vendor specific parameter. Determines whether the corresponding DataReader sends positive acknowledgments for reliability.*
- struct **DDS_EntityNameQosPolicy subscription_name**

  *<<**extension**>> (p. 432) The subscription name and role name.*
- struct **DDS_EndpointTrustProtectionInfo trust_protection_info**

  *<<**extension**>> (p. 432) Trust plugins protection information associated with the discovered DataReader.*
- struct **DDS_EndpointTrustAlgorithmInfo trust_algorithm_info**

*<<**extension**>> (p. 432) Trust Plugins algorithms associated with the discovered DataReader.*

- **DDS_BuiltinTopicKey_t related_datawriter_key**

  *<<**extension**>> (p. 432) Key of the related DataWriter that is created as a part of the Remote Procedure Call (RPC) or Request-Reply endpoints.*

9.181.1 Detailed Description

Entry created when a **DDSDataReader** (p. 1556) is discovered in association with its Subscriber.

Data associated with the built-in topic **DDS_SUBSCRIPTION_TOPIC_NAME** (p. 496). It contains QoS policies and additional information that apply to the remote **DDSDataReader** (p. 1556) the related **DDSSubscriber** (p. 1860).

See also

DDS_SUBSCRIPTION_TOPIC_NAME (p. 496)

DDSSubscriptionBuiltinTopicDataDataReader (p. 1882)

9.181.2 Member Data Documentation

9.181.2.1 key

```
DDS_BuiltinTopicKey_t DDS_SubscriptionBuiltinTopicData::key
```

DCPS key to distinguish entries.

9.181.2.2 participant_key

```
DDS_BuiltinTopicKey_t DDS_SubscriptionBuiltinTopicData::participant_key
```

DCPS key of the participant to which the DataReader belongs.

9.181.2.3 topic_name

```
char* DDS_SubscriptionBuiltinTopicData::topic_name
```

Name of the related **DDSTopic** (p. 1885).

The length of this string is limited to 255 characters.

The memory for this field is managed as described in **String Conventions** (p. 758).

See also

String Conventions (p. 758)

9.181.2.4 type_name

```
char* DDS_SubscriptionBuiltinTopicData::type_name
```

Name of the type attached to the **DDSTopic** (p. 1885).

The length of this string is limited to 255 characters.

The memory for this field is managed as described in **String Conventions** (p. 758).

See also

String Conventions (p. 758)

9.181.2.5 durability

```
struct DDS_DurabilityQosPolicy DDS_SubscriptionBuiltinTopicData::durability
```

Policy of the corresponding DataReader.

9.181.2.6 deadline

```
struct DDS_DeadlineQosPolicy DDS_SubscriptionBuiltinTopicData::deadline
```

Policy of the corresponding DataReader.

9.181.2.7 latency_budget

```
struct DDS_LatencyBudgetQosPolicy DDS_SubscriptionBuiltinTopicData::latency_budget
```

Policy of the corresponding DataReader.

9.181.2.8 liveliness

```
struct DDS_LivelinessQosPolicy DDS_SubscriptionBuiltinTopicData::liveliness
```

Policy of the corresponding DataReader.

9.181.2.9 reliability

`struct` **DDS_ReliabilityQosPolicy** DDS_SubscriptionBuiltinTopicData::reliability

Policy of the corresponding DataReader.

9.181.2.10 ownership

`struct` **DDS_OwnershipQosPolicy** DDS_SubscriptionBuiltinTopicData::ownership

Policy of the corresponding DataReader.

9.181.2.11 destination_order

`struct` **DDS_DestinationOrderQosPolicy** DDS_SubscriptionBuiltinTopicData::destination_order

Policy of the corresponding DataReader.

Warning

Only the field **DDS_DestinationOrderQosPolicy::kind** (p. 933) is propagated during discovery. The other fields always contain their default values.

9.181.2.12 user_data

`struct` **DDS_UserDataQosPolicy** DDS_SubscriptionBuiltinTopicData::user_data

Policy of the corresponding DataReader.

9.181.2.13 time_based_filter

`struct` **DDS_TimeBasedFilterQosPolicy** DDS_SubscriptionBuiltinTopicData::time_based_filter

Policy of the corresponding DataReader.

9.181.2.14 presentation

struct **DDS_PresentationQosPolicy** DDS_SubscriptionBuiltinTopicData::presentation

Policy of the Subscriber to which the DataReader belongs.

9.181.2.15 partition

struct **DDS_PartitionQosPolicy** DDS_SubscriptionBuiltinTopicData::partition

Policy of the Subscriber to which the DataReader belongs.

9.181.2.16 topic_data

struct **DDS_TopicDataQosPolicy** DDS_SubscriptionBuiltinTopicData::topic_data

Policy of the related Topic.

9.181.2.17 group_data

struct **DDS_GroupDataQosPolicy** DDS_SubscriptionBuiltinTopicData::group_data

Policy of the Subscriber to which the DataReader belongs.

9.181.2.18 type_consistency

struct **DDS_TypeConsistencyEnforcementQosPolicy** DDS_SubscriptionBuiltinTopicData::type_consistency

Policy of the corresponding DataReader.

9.181.2.19 representation

struct **DDS_DataRepresentationQosPolicy** DDS_SubscriptionBuiltinTopicData::representation

Data representation policy of the corresponding DataReader.

9.181.2.20 data_tags

DDS_DataTagQosPolicy DDS_SubscriptionBuiltinTopicData::data_tags

Tags of the corresponding DataReader.

9.181.2.21 type_code

struct **DDS_TypeCode*** DDS_SubscriptionBuiltinTopicData::type_code

<<***extension***>> *(p. 432)* Type code information of the corresponding Topic.

If using TypeObject v2 (**DDS_DISCOVERYCONFIG_TYPE_LOOKUP_SERVICE_CHANNEL** (p. 592)) this is NULL if the type is not yet resolved. If the remote endpoint's topic name matches the **DDS_DiscoveryConfigQosPolicy**↩ **::request_types_filter** (p. 959), this will always contain the TypeCode converted from the complete TypeObject v2 representation (if non-NULL); the minimal TypeObject v2 will not be provided even if available. If the topic name does not match the **DDS_DiscoveryConfigQosPolicy::request_types_filter** (p. 959) and the complete TypeObject v2 is resolved, this will contain the TypeCode converted from the complete TypeObject v2 representation. If the topic name does not match the **DDS_DiscoveryConfigQosPolicy::request_types_filter** (p. 959), and only the minimal Type↩ Object v2 is resolved, this will contain the TypeCode converted from the minimal TypeObject v2 representation. As such, information that is not available in the minimal TypeObject v2 representation will be lost (such as default values) and member names will be the stringified version of the hashed member name, rather than the full member name. See `TypeLookup Service` for more information.

9.181.2.22 subscriber_key

DDS_BuiltinTopicKey_t DDS_SubscriptionBuiltinTopicData::subscriber_key

<<***extension***>> *(p. 432)* DCPS key of the subscriber to which the DataReader belongs.

9.181.2.23 property

struct **DDS_PropertyQosPolicy** DDS_SubscriptionBuiltinTopicData::property

<<***extension***>> *(p. 432)* Properties of the corresponding DataReader.

9.181.2.24 unicast_locators

struct **DDS_LocatorSeq** DDS_SubscriptionBuiltinTopicData::unicast_locators

<<***extension***>> *(p. 432)* Custom unicast locators that the endpoint can specify. The default locators will be used if this is not specified.

9.181.2.25 multicast_locators

```
struct DDS_LocatorSeq DDS_SubscriptionBuiltinTopicData::multicast_locators
```

<<***extension***>> *(p. 432)* Custom multicast locators that the endpoint can specify. The default locators will be used if this is not specified.

9.181.2.26 content_filter_property

```
struct DDS_ContentFilterProperty_t DDS_SubscriptionBuiltinTopicData::content_filter_property
```

<<***extension***>> *(p. 432)* This field provides all the required information to enable content filtering on the Writer side.

9.181.2.27 virtual_guid

```
struct DDS_GUID_t DDS_SubscriptionBuiltinTopicData::virtual_guid
```

<<***extension***>> *(p. 432)* Virtual GUID associated to the DataReader.

See also

DDS_GUID_t (p. 1161)

9.181.2.28 service

```
struct DDS_ServiceQosPolicy DDS_SubscriptionBuiltinTopicData::service
```

<<***extension***>> *(p. 432)* Policy of the corresponding DataReader.

9.181.2.29 rtps_protocol_version

```
DDS_ProtocolVersion_t DDS_SubscriptionBuiltinTopicData::rtps_protocol_version
```

<<***extension***>> *(p. 432)* Version number of the RTPS wire protocol used.

9.181.2.30 rtps_vendor_id

```
struct DDS_VendorId_t DDS_SubscriptionBuiltinTopicData::rtps_vendor_id
```

<<***extension***>> *(p. 432)* ID of vendor implementing the RTPS wire protocol.

9.181.2.31 product_version

```
struct DDS_ProductVersion_t DDS_SubscriptionBuiltinTopicData::product_version
```

<<***extension***>> *(p. 432)* This is a vendor specific parameter. It gives the current version of RTI Connext

9.181.2.32 disable_positive_acks

```
DDS_Boolean DDS_SubscriptionBuiltinTopicData::disable_positive_acks
```

<<***extension***>> *(p. 432)* This is a vendor specific parameter. Determines whether the corresponding DataReader sends positive acknowledgments for reliability.

9.181.2.33 subscription_name

```
struct DDS_EntityNameQosPolicy DDS_SubscriptionBuiltinTopicData::subscription_name
```

<<***extension***>> *(p. 432)* The subscription name and role name.

This member contains the name and the role name of the discovered subscription.

9.181.2.34 trust_protection_info

```
struct DDS_EndpointTrustProtectionInfo DDS_SubscriptionBuiltinTopicData::trust_protection_info
```

<<***extension***>> *(p. 432)* Trust plugins protection information associated with the discovered DataReader.

Trust Plugins is a generic abstraction that represents any plugin intended to do transformation, interception, and validation of exchanged data and metadata.

trust_protection_info contains information about how RTPS wire serialization, discovery, and liveliness interact with the loaded Trust Plugins. Two endpoints will not match if their trust_protection_info is incompatible.

The meaning of the contents of this field may vary depending on what Trust Plugins the endpoint is using. For information about how this field interacts with the RTI Security Plugins, please refer to the `RTI Security Plugins User's Manual`.

9.181.2.35 trust_algorithm_info

```
struct DDS_EndpointTrustAlgorithmInfo DDS_SubscriptionBuiltinTopicData::trust_algorithm_info
```

<<***extension***>> *(*p. *432)* Trust Plugins algorithms associated with the discovered DataReader.

Trust Plugins is a generic abstraction that represents any plugin intended to do transformation, interception, and validation of exchanged data and metadata.

trust_algorithm_info contains information about what algorithms the loaded Trust Plugins are running. Two endpoints will not match if their trust_algorithm_info are incompatible.

The meaning of the contents of this field may vary depending on what Trust Plugins the endpoint is using. For information about how this field interacts with the RTI Security Plugins, please refer to the `RTI Security Plugins User's Manual`.

9.181.2.36 related_datawriter_key

```
DDS_BuiltinTopicKey_t DDS_SubscriptionBuiltinTopicData::related_datawriter_key
```

<<***extension***>> *(*p. *432)* Key of the related DataWriter that is created as a part of the Remote Procedure Call (RPC) or Request-Reply endpoints.

This field will be set to something other than **DDS_BUILTIN_TOPIC_KEY_UNKNOWN** (p. 508) only when the discovered DataReader is a part of Remote Procedure Call (RPC) or Request-Reply endpoints.

9.182 DDS_SubscriptionBuiltinTopicDataSeq Struct Reference

Instantiates **FooSeq** (p. 1965) < **DDS_SubscriptionBuiltinTopicData** (p. 1370) > .

9.182.1 Detailed Description

Instantiates **FooSeq** (p. 1965) < **DDS_SubscriptionBuiltinTopicData** (p. 1370) > .

Instantiates:

<<***generic***>> *(*p. *433)* **FooSeq** (p. 1965)

See also

DDS_SubscriptionBuiltinTopicData (p. 1370)

9.183 DDS_SubscriptionMatchedStatus Struct Reference

DDS_SUBSCRIPTION_MATCHED_STATUS (p. 543)

Public Attributes

- **DDS_Long total_count**

  *The total cumulative number of times that this **DDSDataReader** (p. 1556) discovered a "match" with a **DDSDataWriter** (p. 1589).*
- **DDS_Long total_count_change**

  The changes in total_count since the last time the listener was called or the status was read.
- **DDS_Long current_count**

  *The current number of DataWriters with which the **DDSDataReader** (p. 1556) is matched.*
- **DDS_Long current_count_peak**

  *<<**extension**>> (p. 432) Greatest number of DataWriters that matched this DataReader simultaneously.*
- **DDS_Long current_count_change**

  The change in current_count since the last time the listener was called or the status was read.
- **DDS_InstanceHandle_t last_publication_handle**

  *This InstanceHandle can be used to look up which remote **DDSDataWriter** (p. 1589) was the last to cause this Data↩ Reader's status to change, using **DDSDataReader::get_matched_publication_data** (p. 1568).*

9.183.1 Detailed Description

DDS_SUBSCRIPTION_MATCHED_STATUS (p. 543)

A "match" happens when the **DDSDataReader** (p. 1556) finds a **DDSDataWriter** (p. 1589) with the same **DDSTopic** (p. 1885), same or compatible data type, and an offered QoS that is compatible with that requested by the **DDSData↩ Reader** (p. 1556). (For information on compatible data types, see the `Extensible Types Guide`.)

This status is also changed (and the listener, if any, called) when a match is ended. A local **DDSDataReader** (p. 1556) will become "unmatched" from a remote **DDSDataWriter** (p. 1589) when that **DDSDataWriter** (p. 1589) goes away for any of the following reasons:

- The **DDSDomainParticipant** (p. 1619) containing the matched **DDSDataWriter** (p. 1589) has lost liveliness.
- This DataReader or the matched DataWriter has changed QoS such that the entities are now incompatible.
- The matched DataWriter has been deleted.

This status may reflect changes from multiple match or unmatch events, and the **DDS_SubscriptionMatchedStatus↩ ::current_count_change** (p. 1381) can be used to determine the number of changes since the listener was called back or the status was checked.

Note: A DataWriter's loss of liveliness (which is determined by **DDS_LivelinessQosPolicyKind** (p. 607)) does not trigger an unmatch event. So a DataWriter may still match even though its liveliness is lost.

9.183.2 Member Data Documentation

9.183.2.1 total_count

DDS_Long DDS_SubscriptionMatchedStatus::total_count

The total cumulative number of times that this **DDSDataReader** (p. 1556) discovered a "match" with a **DDSDataWriter** (p. 1589).

This number increases whenever a new match is discovered. It does not decrease when an existing match goes away for any of the reasons described in **DDS_SubscriptionMatchedStatus** (p. 1379).

9.183.2.2 total_count_change

DDS_Long DDS_SubscriptionMatchedStatus::total_count_change

The changes in total_count since the last time the listener was called or the status was read.

Note that this number will never be negative (because it's the total number of times the DataReader ever matched with a DataWriter).

9.183.2.3 current_count

DDS_Long DDS_SubscriptionMatchedStatus::current_count

The current number of DataWriters with which the **DDSDataReader** (p. 1556) is matched.

This number increases when a new match is discovered and decreases when an existing match goes away for any of the reasons described in **DDS_SubscriptionMatchedStatus** (p. 1379).

9.183.2.4 current_count_peak

DDS_Long DDS_SubscriptionMatchedStatus::current_count_peak

<<***extension***>> *(p. 432)* Greatest number of DataWriters that matched this DataReader simultaneously.

That is, there was no moment in time when more than this many DataWriters matched this DataReader. (As a result, total_count can be higher than current_count_peak.)

9.183.2.5 current_count_change

DDS_Long DDS_SubscriptionMatchedStatus::current_count_change

The change in current_count since the last time the listener was called or the status was read.

Note that a negative current_count_change means that one or more DataWriters have become unmatched for one or more of the reasons described in **DDS_SubscriptionMatchedStatus** (p. 1379).

9.183.2.6 last_publication_handle

```
DDS_InstanceHandle_t DDS_SubscriptionMatchedStatus::last_publication_handle
```

This InstanceHandle can be used to look up which remote **DDSDataWriter** (p. 1589) was the last to cause this DataReader's status to change, using **DDSDataReader::get_matched_publication_data** (p. 1568).

If the DataWriter no longer matches this DataReader due to any of the reasons in **DDS_SubscriptionMatchedStatus** (p. 1379) except incompatible QoS, then the DataWriter has been purged from this DataReader's DomainParticipant discovery database. (See the "Discovery Overview" section of the `User's Manual`.) In that case, the **DDSDataReader::get_matched_publication_data** (p. 1568) method will not be able to return information about the DataWriter. The only way to get information about the lost DataWriter is if you cached the information previously.

9.184 DDS_SystemResourceLimitsQosPolicy Struct Reference

*<<**extension**>> (p. 213)* Configures **DDSDomainParticipant** (p. 1619)-independent resources used by RTI Connext. Mainly used to change the maximum number of **DDSDomainParticipant** (p. 1619) entities that can be created within a single process (address space).

Public Attributes

- **DDS_Long max_objects_per_thread**

  [DEPRECATED] *(p. **384**) <<**extension**>> (p. 213) The maximum number of objects that can be stored per thread for a **DDSDomainParticipantFactory** (p. 1694).*
- **DDS_Long initial_objects_per_thread**

  *<<**extension**>> (p. 213) The number of objects per thread for a **DDSDomainParticipantFactory** (p. 1694) for which infrastructure will initially be allocated.*

9.184.1 Detailed Description

*<<**extension**>> (p. 213)* Configures **DDSDomainParticipant** (p. 1619)-independent resources used by RTI Connext. Mainly used to change the maximum number of **DDSDomainParticipant** (p. 1619) entities that can be created within a single process (address space).

Entity:

DDSDomainParticipantFactory (p. 1694)

Properties:

RxO (p. 556) = N/A
Changeable (p. 556) = **NO** (p. 556)

9.184.2 Usage

Within a single process (or address space for some supported real-time operating systems), applications may create and use multiple **DDSDomainParticipant** (p. 1619) entities. This QoS policy sets a parameter that places an effective upper bound on the maximum number of **DDSDomainParticipant** (p. 1619) entities that can be created in a single process/address space. These values cannot be changed after a DomainParticipant has been created and they cannot be set in a QoS XML file.

9.184.3 Member Data Documentation

9.184.3.1 max_objects_per_thread

```
DDS_Long DDS_SystemResourceLimitsQosPolicy::max_objects_per_thread
```

[DEPRECATED] (p. **384)** <<***extension***>> *(*p. *213)* The maximum number of objects that can be stored per thread for a **DDSDomainParticipantFactory** (p. 1694).

This value is the upper bound on the number of objects that can be stored per thread. When a **DDSDomain↩ ParticipantFactory** (p. 1694) is created, infrastructure will be created to manage the number of objects specified by initial_objects_per_thread. As more objects are required by the application, the infrastructure will be automatically grown to accommodate up to max_objects_per_thread objects. Leave this property set to the default value to allow the infrastructure to grow as needed. If you wish to strictly control memory allocation, set max_objects_per_thread to a smaller value, but note that this runs the risk of a runtime error and reduced application functionality if your limit is reached.

[default] 261120

[range] [1, 261120]

Categories

Immutable (p. 333), **Middleware Memory Management** (p. 297)

9.184.3.2 initial_objects_per_thread

```
DDS_Long DDS_SystemResourceLimitsQosPolicy::initial_objects_per_thread
```

<<***extension***>> *(p. 213)* The number of objects per thread for a **DDSDomainParticipantFactory** (p. 1694) for which infrastructure will initially be allocated.

The infrastructure for managing thread-specific objects will initially be sized according to this value. The infrastructure will grow automatically, up to a maximum of max_objects_per_thread, as required by the application at runtime. If you are certain that more than the default value of initial_objects_per_thread will be required for your application and you wish to reduce the number of memory allocations performed while your application reaches steady state, you may set this value to a larger number. To improve the efficiency of memory allocation, RTI Connext may initially size the infrastructure to a larger value than the value of initial_objects_per_thread. The infrastructure will never be sized less than initial_objects_per_thread or greater than max_objects_per_thread. When the infrastructure for managing thread-specific objects is created or increased, a log message stating "Allowed number of thread specific objects is now " will be produced at the local log level.

[default] 1024

[range] [1, 261120]; must be less than or equal to max_objects_per_thread

Categories

Immutable (p. 333), **Middleware Memory Management** (p. 297)

9.185 DDS_Tag Struct Reference

Tags are name/value pair objects.

Public Attributes

- char * **name**

  Tag name.
- char * **value**

  Tag value.

9.185.1 Detailed Description

Tags are name/value pair objects.

9.185.2 Member Data Documentation

9.185.2.1 name

```
char* DDS_Tag::name
```

Tag name.

It must be a NULL-terminated string.

9.185.2.2 value

```
char* DDS_Tag::value
```

Tag value.

It must be a NULL-terminated string.

9.186 DDS_TagSeq Struct Reference

Declares IDL `sequence` < **DDS_Tag** (p. 1384) >

9.186.1 Detailed Description

Declares IDL `sequence` < **DDS_Tag** (p. 1384) >

See also

DDS_Tag (p. 1384)

9.187 DDS_ThreadSettings_t Struct Reference

The properties of a thread of execution.

Public Attributes

- **DDS_ThreadSettingsKindMask mask**
  Describes the type of thread.
- **DDS_Long priority**
  Thread priority.
- **DDS_Long stack_size**
  The thread stack-size.
- struct **DDS_LongSeq cpu_list**
  The list of processors on which the thread(s) may run.
- **DDS_ThreadSettingsCpuRotationKind cpu_rotation**
  Determines how processor affinity is applied to multiple threads.

9.187.1 Detailed Description

The properties of a thread of execution.

QoS:

DDS_EventQosPolicy (p. 1150) **DDS_DatabaseQosPolicy** (p. 831) **DDS_ReceiverPoolQosPolicy** (p. 1293) **DDS_AsynchronousPublisherQosPolicy** (p. 798)

9.187.2 Member Data Documentation

9.187.2.1 mask

DDS_ThreadSettingsKindMask DDS_ThreadSettings_t::mask

Describes the type of thread.

The meaning of each bit of the mask are defined by **DDS_ThreadSettingsKind** (p. 548).

[default] 0, use default options of the OS

9.187.2.2 priority

DDS_Long DDS_ThreadSettings_t::priority

Thread priority.

[range] Platform-dependent - Consult `Platform Notes` for additional details.

9.187.2.3 stack_size

DDS_Long DDS_ThreadSettings_t::stack_size

The thread stack-size.

[range] Platform-dependent. Consult `Platform Notes` for additional details.

Categories

Middleware Memory Management (p. 297)

9.187.2.4 cpu_list

```
struct DDS_LongSeq DDS_ThreadSettings_t::cpu_list
```

The list of processors on which the thread(s) may run.

A sequence of integers that represent the set of processors on which the thread(s) controlled by this QoS may run. An empty sequence (the default) means the middleware wlll make no CPU affinity adjustments.

Note: This feature is currently only supported on a subset of architectures (see the `Platform Notes`). The API may change as more architectures are added in future releases.

This value is only relevant to the **DDS_ReceiverPoolQosPolicy** (p. 1293). It is ignored within other QoS policies that include **DDS_ThreadSettings_t** (p. 1385).

See also

Controlling CPU Core Affinity for RTI Threads (p. 548)

[default] Empty sequence

9.187.2.5 cpu_rotation

```
DDS_ThreadSettingsCpuRotationKind DDS_ThreadSettings_t::cpu_rotation
```

Determines how processor affinity is applied to multiple threads.

This value is only relevant to the **DDS_ReceiverPoolQosPolicy** (p. 1293). It is ignored within other QoS policies that include **DDS_ThreadSettings_t** (p. 1385).

See also

Controlling CPU Core Affinity for RTI Threads (p. 548)

Note: This feature is currently only supported on a subset of architectures (see the `Platform Notes`). The API may change as more architectures are added in future releases. ;

9.188 DDS_Time_t Struct Reference

Type for *time* representation.

Static Public Member Functions

- static **DDS_Time_t from_micros** (**DDS_UnsignedLongLong** microseconds)
  Creates a new time object from a time expressed in microseconds.
- static **DDS_Time_t from_millis** (**DDS_UnsignedLongLong** milliseconds)
  Creates a new time object from a time expressed in milliseconds.
- static **DDS_Time_t from_nanos** (**DDS_UnsignedLongLong** nanoseconds)
  Creates a new time object from a time expressed in nanoseconds.
- static **DDS_Time_t from_seconds** (**DDS_UnsignedLongLong** seconds)
  Creates a new time object from a time expressed in seconds.

Public Attributes

- **DDS_LongLong sec**
  seconds
- **DDS_UnsignedLong nanosec**
  nanoseconds

9.188.1 Detailed Description

Type for *time* representation.

A **DDS_Time_t** (p. 1387) represents a moment in time.

9.188.2 Member Data Documentation

9.188.2.1 sec

```
DDS_LongLong DDS_Time_t::sec
```

seconds

9.188.2.2 nanosec

```
DDS_UnsignedLong DDS_Time_t::nanosec
```

nanoseconds

[range] [0,1000000000)

9.189 DDS_TimeBasedFilterQosPolicy Struct Reference

<<***spec***>> *(p. 209)* Filter that allows a **DDSDataReader** (p. 1556) to specify that it is interested only in (potentially) a subset of the values of the data.

Public Attributes

- struct **DDS_Duration_t minimum_separation**

  <<***basic***>> *(p. 201)* <<***spec***>> *(p. 209) The minimum separation duration between subsequent samples.*

9.189.1 Detailed Description

<<***spec***>> *(p. 209)* Filter that allows a **DDSDataReader** (p. 1556) to specify that it is interested only in (potentially) a subset of the values of the data.

The filter states that the **DDSDataReader** (p. 1556) does not want to receive more than one value each `minimum_separation`, regardless of how fast the changes occur.

Entity:

DDSDataReader (p. 1556)

Properties:

RxO (p. 556) = N/A
Changeable (p. 556) = **YES** (p. 556)

9.189.2 Usage

You can use this QoS policy to reduce the amount of data received by a **DDSDataReader** (p. 1556). **DDSDataWriter** (p. 1589) entities may send data faster than needed by a **DDSDataReader** (p. 1556). For example, a **DDSDataReader** (p. 1556) of sensor data that is displayed to a human operator in a GUI application does not need to receive data updates faster than a user can reasonably perceive changes in data values. This is often measured in tenths (0.1) of a second up to several seconds. However, a **DDSDataWriter** (p. 1589) of sensor information may have other **DDSDataReader** (p. 1556) entities that are processing the sensor information to control parts of the system and thus need new data updates in measures of hundredths (0.01) or thousandths (0.001) of a second.

With this QoS policy, different **DDSDataReader** (p. 1556) entities can set their own time-based filters, so that data published faster than the period set by a each **DDSDataReader** (p. 1556) will not be delivered to that **DDSDataReader** (p. 1556).

The **TIME_BASED_FILTER** (p. 639) also applies to each instance separately; that is, the constraint is that the **DDSDataReader** (p. 1556) does not want to see more than one sample of each instance per `minimum_separation` period.

This QoS policy allows you to optimize resource usage (CPU and possibly network bandwidth) by only delivering the required amount of data to each **DDSDataReader** (p. 1556), accommodating the fact that, for rapidly-changing data, different subscribers may have different requirements and constraints as to how frequently they need or can handle being notified of the most current values. As such, it can also be used to protect applications that are running on a heterogeneous network where some nodes are capable of generating data much faster than others can consume it.

For best effort data delivery, if the data type is unkeyed and the **DDSDataWriter** (p. 1589) has an infinite **DDS_↩ LivelinessQosPolicy::lease_duration** (p. 1182), RTI Connext will only send as many packets to a **DDSDataReader** (p. 1556) as required by the TIME_BASED_FILTER, no matter how fast **FooDataWriter::write** (p. 1951) is called.

For multicast data delivery to multiple DataReaders, the one with the lowest `minimum_separation` determines the DataWriter's send rate. For example, if a **DDSDataWriter** (p. 1589) sends over multicast to two DataReaders, one with `minimum_separation` of 2 seconds and one with `minimum_separation` of 1 second, the DataWriter will send every 1 second.

In configurations where RTI Connext must send all the data published by the **DDSDataWriter** (p. 1589) (for example, when the **DDSDataWriter** (p. 1589) is reliable, when the data type is keyed, or when the **DDSDataWriter** (p. 1589) has a finite **DDS_LivelinessQosPolicy::lease_duration** (p. 1182)), only the data that passes the TIME_BASED_FILTER will be stored in the receive queue of the **DDSDataReader** (p. 1556). Extra data will be accepted but dropped. Note that filtering is only applied on alive samples (that is, samples that have not been disposed/unregistered).

9.189.3 Consistency

It is inconsistent for a **DDSDataReader** (p. 1556) to have a `minimum_separation` longer than its **DEADLINE** (p. 583) `period`.

However, it is important to be aware of certain edge cases that can occur when your publication rate, minimum separation, and deadline period align and that can cause missed deadlines that you may not expect. For example, suppose that you nominally publish samples every second but that this rate can vary somewhat over time. You declare a minimum separation of 1 second to filter out rapid updates and set a deadline of two seconds so that you will be aware if the rate falls too low. Even if your update rate never wavers, you can still miss deadlines! Here's why:

Suppose you publish the first sample at time t=0 seconds. You then publish your next sample at t=1 seconds. Depending on how your operating system schedules the time-based filter execution relative to the publication, this second sample may be filtered. You then publish your third sample at t=2 seconds, and depending on how your OS schedules this publication in relation to the deadline check, you could miss the deadline.

This scenario demonstrates a couple of rules of thumb:

- Beware of setting your `minimum_separation` to a value very close to your publication rate: you may filter more data than you intend to.
- Beware of setting your `minimum_separation` to a value that is too close to your deadline period relative to your publication rate. You may miss deadlines.

See **DDS_DeadlineQosPolicy** (p. 928) for more information about the interactions between deadlines and time-based filters.

The setting of a **TIME_BASED_FILTER** (p. 639) – that is, the selection of a `minimum_separation` with a value greater than zero – is consistent with all settings of the **HISTORY** (p. 603) and **RELIABILITY** (p. 631) QoS. The **TIME↩ _BASED_FILTER** (p. 639) specifies the samples that are of interest to the **DDSDataReader** (p. 1556). The **HISTORY** (p. 603) and **RELIABILITY** (p. 631) QoS affect the behavior of the middleware with respect to the samples that have been determined to be of interest to the **DDSDataReader** (p. 1556); that is, they apply *after* the **TIME_BASED_FILTER** (p. 639) has been applied.

In the case where the reliability QoS kind is **DDS_RELIABLE_RELIABILITY_QOS** (p. 633), in steady-state – defined as the situation where the **DDSDataWriter** (p. 1589) does not write new samples for a period "long" compared to the `minimum_separation` – the system should guarantee delivery of the last sample to the **DDSDataReader** (p. 1556).

See also

DeadlineQosPolicy

HistoryQosPolicy

ReliabilityQosPolicy

9.189.4 Member Data Documentation

9.189.4.1 minimum_separation

`struct` **DDS_Duration_t** DDS_TimeBasedFilterQosPolicy::minimum_separation

<<***basic***>> *(*p. *201)* <<***spec***>> *(*p. *209)* The minimum separation duration between subsequent samples.

[default] 0 (meaning the **DDSDataReader** (p. 1556) is potentially interested in all values)

[range] [0,1 year], < **DDS_DeadlineQosPolicy::period** (p. 930)

Categories

Mutable (p. 369), **Discovery Metadata** (p. 377), **Applied Per-Instance** (p. 375), **Data Availability** (p. 261)

9.190 DDS_TopicBuiltinTopicData Struct Reference

Entry created when a Topic object discovered.

Public Attributes

- **DDS_BuiltinTopicKey_t key**

  DCPS key to distinguish entries.
- char * **name**

  Name of the ***DDSTopic*** *(*p. *1885).*
- char * **type_name**

  Name of the type attached to the ***DDSTopic*** *(*p. *1885).*
- struct **DDS_DurabilityQosPolicy durability**

  durability policy of the corresponding Topic
- struct **DDS_DurabilityServiceQosPolicy durability_service**

  durability service policy of the corresponding Topic
- struct **DDS_DeadlineQosPolicy deadline**

  Policy of the corresponding Topic.
- struct **DDS_LatencyBudgetQosPolicy latency_budget**

  Policy of the corresponding Topic.

- struct **DDS_LivelinessQosPolicy liveliness**
  Policy of the corresponding Topic.
- struct **DDS_ReliabilityQosPolicy reliability**
  Policy of the corresponding Topic.
- struct **DDS_TransportPriorityQosPolicy transport_priority**
  Policy of the corresponding Topic.
- struct **DDS_LifespanQosPolicy lifespan**
  Policy of the corresponding Topic.
- struct **DDS_DestinationOrderQosPolicy destination_order**
  Policy of the corresponding Topic.
- struct **DDS_HistoryQosPolicy history**
  Policy of the corresponding Topic.
- struct **DDS_ResourceLimitsQosPolicy resource_limits**
  Policy of the corresponding Topic.
- struct **DDS_OwnershipQosPolicy ownership**
  Policy of the corresponding Topic.
- struct **DDS_TopicDataQosPolicy topic_data**
  Policy of the corresponding Topic.
- struct **DDS_DataRepresentationQosPolicy representation**
  Data representation policy of the corresponding Topic.

9.190.1 Detailed Description

Entry created when a Topic object discovered.

Data associated with the built-in topic **DDS_TOPIC_TOPIC_NAME** (p. 493). It contains QoS policies and additional information that apply to the remote **DDSTopic** (p. 1885).

Note: The **DDS_TopicBuiltinTopicData** (p. 1391) built-in topic is meant to convey information about discovered Topics. This Topic's samples are not propagated in a separate packet on the wire. Instead, the data is sent as part of the information carried by other built-in topics (**DDS_PublicationBuiltinTopicData** (p. 1264) and **DDS_Subscription↩ BuiltinTopicData** (p. 1370)). Therefore TopicBuiltinTopicData DataReaders will not receive any data.

See also

DDS_TOPIC_TOPIC_NAME (p. 493)

DDSTopicBuiltinTopicDataDataReader (p. 1890)

9.190.2 Member Data Documentation

9.190.2.1 key

```
DDS_BuiltinTopicKey_t DDS_TopicBuiltinTopicData::key
```

DCPS key to distinguish entries.

9.190.2.2 name

```
char* DDS_TopicBuiltinTopicData::name
```

Name of the **DDSTopic** (p. 1885).

The length of this string is limited to 255 characters.

The memory for this field is managed as described in **String Conventions** (p. 758).

See also

String Conventions (p. 758)

9.190.2.3 type_name

```
char* DDS_TopicBuiltinTopicData::type_name
```

Name of the type attached to the **DDSTopic** (p. 1885).

The length of this string is limited to 255 characters.

The memory for this field is managed as described in **String Conventions** (p. 758).

See also

String Conventions (p. 758)

9.190.2.4 durability

```
struct DDS_DurabilityQosPolicy DDS_TopicBuiltinTopicData::durability
```

durability policy of the corresponding Topic

9.190.2.5 durability_service

```
struct DDS_DurabilityServiceQosPolicy DDS_TopicBuiltinTopicData::durability_service
```

durability service policy of the corresponding Topic

9.190.2.6 deadline

`struct` **DDS_DeadlineQosPolicy** DDS_TopicBuiltinTopicData::deadline

Policy of the corresponding Topic.

9.190.2.7 latency_budget

`struct` **DDS_LatencyBudgetQosPolicy** DDS_TopicBuiltinTopicData::latency_budget

Policy of the corresponding Topic.

9.190.2.8 liveliness

`struct` **DDS_LivelinessQosPolicy** DDS_TopicBuiltinTopicData::liveliness

Policy of the corresponding Topic.

9.190.2.9 reliability

`struct` **DDS_ReliabilityQosPolicy** DDS_TopicBuiltinTopicData::reliability

Policy of the corresponding Topic.

9.190.2.10 transport_priority

`struct` **DDS_TransportPriorityQosPolicy** DDS_TopicBuiltinTopicData::transport_priority

Policy of the corresponding Topic.

9.190.2.11 lifespan

`struct` **DDS_LifespanQosPolicy** DDS_TopicBuiltinTopicData::lifespan

Policy of the corresponding Topic.

9.190.2.12 destination_order

struct **DDS_DestinationOrderQosPolicy** DDS_TopicBuiltinTopicData::destination_order

Policy of the corresponding Topic.

9.190.2.13 history

struct **DDS_HistoryQosPolicy** DDS_TopicBuiltinTopicData::history

Policy of the corresponding Topic.

9.190.2.14 resource_limits

struct **DDS_ResourceLimitsQosPolicy** DDS_TopicBuiltinTopicData::resource_limits

Policy of the corresponding Topic.

9.190.2.15 ownership

struct **DDS_OwnershipQosPolicy** DDS_TopicBuiltinTopicData::ownership

Policy of the corresponding Topic.

9.190.2.16 topic_data

struct **DDS_TopicDataQosPolicy** DDS_TopicBuiltinTopicData::topic_data

Policy of the corresponding Topic.

9.190.2.17 representation

struct **DDS_DataRepresentationQosPolicy** DDS_TopicBuiltinTopicData::representation

Data representation policy of the corresponding Topic.

9.191 DDS_TopicBuiltinTopicDataSeq Struct Reference

Instantiates **FooSeq** (p. 1965) < **DDS_TopicBuiltinTopicData** (p. 1391) > .

9.191.1 Detailed Description

Instantiates **FooSeq** (p. 1965) < **DDS_TopicBuiltinTopicData** (p. 1391) > .

Instantiates:

<<***generic***>> *(*p. *433)* **FooSeq** (p. 1965)

See also

DDS_TopicBuiltinTopicData (p. 1391)

9.192 DDS_TopicDataQosPolicy Struct Reference

Attaches a buffer of opaque data that is distributed by means of **Built-in Topics** (p. 61) during discovery.

Public Attributes

- struct **DDS_OctetSeq value**
  a sequence of octets

9.192.1 Detailed Description

Attaches a buffer of opaque data that is distributed by means of **Built-in Topics** (p. 61) during discovery.

Entity:

DDSTopic (p. 1885)

Properties:

RxO (p. 556) = NO
Changeable (p. 556) = **YES** (p. 556)

See also

DDSDomainParticipant::get_builtin_subscriber (p. 1661)

9.192.2 Usage

The purpose of this QoS is to allow the application to attach additional information to the created **DDSTopic** (p. 1885) objects, so that when a remote application discovers their existence, it can access that information and use it for its own purposes. This extra data is not used by RTI Connext.

One possible use of this QoS is to attach security credentials or some other information that can be used by the remote application to authenticate the source.

In combination with **DDSDataReaderListener** (p. 1583), **DDSDataWriterListener** (p. 1612), or operations such as **DDSDomainParticipant::ignore_topic** (p. 1663), this QoS policy can assist an application in defining and enforcing its own security policies.

The use of this QoS is not limited to security; it offers a simple, yet flexible extensibility mechanism.

Important: RTI Connext stores the data placed in this policy in pre-allocated pools. It is therefore necessary to configure RTI Connext with the maximum size of the data that will be stored in policies of this type. This size is configured with **DDS_DomainParticipantResourceLimitsQosPolicy::topic_data_max_length** (p. 991).

9.192.3 Member Data Documentation

9.192.3.1 value

`struct` **`DDS_OctetSeq`** `DDS_TopicDataQosPolicy::value`

a sequence of octets

[default] empty (zero-length)

[range] Octet sequence of length [0,max_length]

9.193 DDS_TopicQos Struct Reference

QoS policies supported by a **DDSTopic** (p. 1885) entity.

Public Member Functions

- bool **operator==** (const **DDS_TopicQos** &r) const

  Compares two TopicQos objects for equality.
- bool **operator!=** (const **DDS_TopicQos** &r) const

  Compares two TopicQos objects for inequality.
- **DDS_ReturnCode_t print** () const

  *Prints this **DDS_TopicQos*** (p. *1397*) *to stdout.*
- **DDS_ReturnCode_t to_string** (char ∗string, **DDS_UnsignedLong** &string_size) const

  *Obtains a string representation of this **DDS_TopicQos*** (p. *1397*).
- **DDS_ReturnCode_t to_string** (char ∗string, **DDS_UnsignedLong** &string_size, const **DDS_TopicQos** &base) const

  *Obtains a string representation of this **DDS_TopicQos*** (p. *1397*).
- **DDS_ReturnCode_t to_string** (char ∗string, **DDS_UnsignedLong** &string_size, const **DDS_QosPrintFormat** &format) const

  *Obtains a string representation of this **DDS_TopicQos*** (p. *1397*).
- **DDS_ReturnCode_t to_string** (char ∗string, **DDS_UnsignedLong** &string_size, const **DDS_TopicQos** &base, const **DDS_QosPrintFormat** &format) const

  *Obtains a string representation of this **DDS_TopicQos*** (p. *1397*).
- **DDS_ReturnCode_t to_string** (char ∗string, **DDS_UnsignedLong** &string_size, const **DDS_QosPrintAll_t** &) const

  *Obtains a string representation of this **DDS_TopicQos*** (p. *1397*).
- **DDS_ReturnCode_t to_string** (char ∗string, **DDS_UnsignedLong** &string_size, const **DDS_QosPrintAll_t** &, const **DDS_QosPrintFormat** &format) const

  *Obtains a string representation of this **DDS_TopicQos*** (p. *1397*).

Public Attributes

- struct **DDS_TopicDataQosPolicy topic_data**

  *Topic data policy, **TOPIC_DATA*** (p. *639*).
- struct **DDS_DurabilityQosPolicy durability**

  *Durability policy, **DURABILITY*** (p. *596*).
- struct **DDS_DurabilityServiceQosPolicy durability_service**

  *DurabilityService policy, **DURABILITY_SERVICE*** (p. *600*).
- struct **DDS_DeadlineQosPolicy deadline**

  *<<**basic**>>* (p. *201*) *Deadline policy, **DEADLINE*** (p. *583*).
- struct **DDS_LatencyBudgetQosPolicy latency_budget**

  *Latency budget policy, **LATENCY_BUDGET*** (p. *605*).
- struct **DDS_LivelinessQosPolicy liveliness**

  *Liveliness policy, **LIVELINESS*** (p. *607*).
- struct **DDS_ReliabilityQosPolicy reliability**

  *Reliability policy, **RELIABILITY*** (p. *631*).
- struct **DDS_DestinationOrderQosPolicy destination_order**

  *Destination order policy, **DESTINATION_ORDER*** (p. *584*).
- struct **DDS_HistoryQosPolicy history**

  *<<**basic**>>* (p. *201*) *History policy, **HISTORY*** (p. *603*).
- struct **DDS_ResourceLimitsQosPolicy resource_limits**

*Resource limits policy, **RESOURCE_LIMITS** (p. 635).*

- struct **DDS_TransportPriorityQosPolicy transport_priority**

  *Transport priority policy, **TRANSPORT_PRIORITY** (p. 647).*
- struct **DDS_LifespanQosPolicy lifespan**

  *<<**basic**>> (p. 201) Lifespan policy, **LIFESPAN** (p. 606).*
- struct **DDS_OwnershipQosPolicy ownership**

  *<<**basic**>> (p. 201) Ownership policy, **OWNERSHIP** (p. 612).*
- struct **DDS_DataRepresentationQosPolicy representation**

  *Data representation policy, **DATA_REPRESENTATION** (p. 567).*

9.193.1 Detailed Description

QoS policies supported by a **DDSTopic** (p. 1885) entity.

You must set certain members in a consistent manner:

length of **DDS_TopicQos::topic_data** (p. 1400) .value <= **DDS_DomainParticipantQos::resource_limits** (p. 974) .topic_data_max_length

If any of the above are not true, **DDSTopic::set_qos** (p. 1887), **DDSTopic::set_qos_with_profile** (p. 1888) and **DDSDomainParticipant::set_default_topic_qos** (p. 1637) will fail with **DDS_RETCODE_INCONSISTENT_POLICY** (p. 533) and **DDSDomainParticipant::create_topic** (p. 1650) will return NULL.

Entity:

DDSTopic (p. 1885)

See also

All QoS Policies (p. 549) allowed ranges within each Qos.

9.193.2 Member Function Documentation

9.193.2.1 operator==()

```
bool DDS_TopicQos::operator== (
            const  DDS_TopicQos & r ) const  [inline]
```

Compares two TopicQos objects for equality.

See also

DDS_TopicQos_equals (p. 67)

References **DDS_TopicQos_equals()**.

9.193.2.2 operator"!=()

```
bool DDS_TopicQos::operator!= (
            const  DDS_TopicQos & r ) const  [inline]
```

Compares two TopicQos objects for inequality.

See also

DDS_TopicQos_equals (p. 67)

References **DDS_TopicQos_equals()**.

9.193.3 Member Data Documentation

9.193.3.1 topic_data

```
struct  DDS_TopicDataQosPolicy DDS_TopicQos::topic_data
```

Topic data policy, **TOPIC_DATA** (p. 639).

9.193.3.2 durability

```
struct  DDS_DurabilityQosPolicy DDS_TopicQos::durability
```

Durability policy, **DURABILITY** (p. 596).

9.193.3.3 durability_service

```
struct  DDS_DurabilityServiceQosPolicy DDS_TopicQos::durability_service
```

DurabilityService policy, **DURABILITY_SERVICE** (p. 600).

9.193.3.4 deadline

`struct` **DDS_DeadlineQosPolicy** DDS_TopicQos::deadline

<<***basic***>> *(p. 201)* Deadline policy, **DEADLINE** (p. 583).

9.193.3.5 latency_budget

`struct` **DDS_LatencyBudgetQosPolicy** DDS_TopicQos::latency_budget

Latency budget policy, **LATENCY_BUDGET** (p. 605).

9.193.3.6 liveliness

`struct` **DDS_LivelinessQosPolicy** DDS_TopicQos::liveliness

Liveliness policy, **LIVELINESS** (p. 607).

9.193.3.7 reliability

`struct` **DDS_ReliabilityQosPolicy** DDS_TopicQos::reliability

Reliability policy, **RELIABILITY** (p. 631).

9.193.3.8 destination_order

`struct` **DDS_DestinationOrderQosPolicy** DDS_TopicQos::destination_order

Destination order policy, **DESTINATION_ORDER** (p. 584).

9.193.3.9 history

`struct` **DDS_HistoryQosPolicy** DDS_TopicQos::history

<<***basic***>> *(p. 201)* History policy, **HISTORY** (p. 603).

9.193.3.10 resource_limits

`struct` **DDS_ResourceLimitsQosPolicy** `DDS_TopicQos::resource_limits`

Resource limits policy, **RESOURCE_LIMITS** (p. 635).

9.193.3.11 transport_priority

`struct` **DDS_TransportPriorityQosPolicy** `DDS_TopicQos::transport_priority`

Transport priority policy, **TRANSPORT_PRIORITY** (p. 647).

9.193.3.12 lifespan

`struct` **DDS_LifespanQosPolicy** `DDS_TopicQos::lifespan`

<<***basic***>> *(p. 201)* Lifespan policy, **LIFESPAN** (p. 606).

9.193.3.13 ownership

`struct` **DDS_OwnershipQosPolicy** `DDS_TopicQos::ownership`

<<***basic***>> *(p. 201)* Ownership policy, **OWNERSHIP** (p. 612).

9.193.3.14 representation

`struct` **DDS_DataRepresentationQosPolicy** `DDS_TopicQos::representation`

Data representation policy, **DATA_REPRESENTATION** (p. 567).

9.194 DDS_TopicQueryData Struct Reference

<<***extension***>> *(p. 432)* Provides information about a **DDSTopicQuery** (p. 1895)

Public Attributes

- struct **DDS_TopicQuerySelection topic_query_selection**

  The data selection.
- char * **topic_name**

  *The topic name of the **DDSDataReader** (p. 1556).*
- struct **DDS_GUID_t original_related_reader_guid**

  *Identifies the **DDSDataReader** (p. 1556) that created the **DDSTopicQuery** (p. 1895).*

9.194.1 Detailed Description

<<***extension***>> *(p. 432)* Provides information about a **DDSTopicQuery** (p. 1895)

Contains the information about a TopicQuery that can be retrieved using **DDSTopicQueryHelper::topic_query_data_from_service_request** (p. 1896).

See also

DDSTopicQueryHelper::topic_query_data_from_service_request (p. 1896)

9.194.2 Member Data Documentation

9.194.2.1 topic_query_selection

```
struct DDS_TopicQuerySelection DDS_TopicQueryData::topic_query_selection
```

The data selection.

9.194.2.2 topic_name

```
char* DDS_TopicQueryData::topic_name
```

The topic name of the **DDSDataReader** (p. 1556).

9.194.2.3 original_related_reader_guid

```
struct DDS_GUID_t DDS_TopicQueryData::original_related_reader_guid
```

Identifies the **DDSDataReader** (p. 1556) that created the **DDSTopicQuery** (p. 1895).

9.195 DDS_TopicQueryDispatchQosPolicy Struct Reference

*<<**extension**>> (p. 213)* Configures the ability of a **DDSDataWriter** (p. 1589) to publish samples in response to a **DDSTopicQuery** (p. 1895)

Public Attributes

- **DDS_Boolean enable**

  *<<**extension**>> (p. 213) Allows this writer to dispatch TopicQueries*
- struct **DDS_Duration_t publication_period**

  *<<**basic**>> (p. 201) <<**extension**>> (p. 213) Sets the periodic interval at which samples are published.*
- **DDS_Long samples_per_period**

  *<<**extension**>> (p. 213) Sets the maximum number of samples to publish in each publication_period.*

9.195.1 Detailed Description

*<<**extension**>> (p. 213)* Configures the ability of a **DDSDataWriter** (p. 1589) to publish samples in response to a **DDSTopicQuery** (p. 1895)

Enables the ability of a **DDSDataWriter** (p. 1589) to publish historical samples upon reception of a **DDSTopicQuery** (p. 1895) and how often they are published.

Since a TopicQuery selects previously written samples, the DataWriter must have a **DDS_DurabilityQosPolicy::kind** (p. 1007) different from **DDS_VOLATILE_DURABILITY_QOS** (p. 598). Also, **DDS_ReliabilityQosPolicy::kind** (p. 1298) must be set to **DDS_RELIABLE_RELIABILITY_QOS** (p. 633).

Only samples that fall within the **DDS_DurabilityQosPolicy::writer_depth** (p. 1008) for an instance are evaluated against the TopicQuery filter. While the DataWriter is waiting for acknowledgements from one or more DataReaders, there may temporarily be more than **DDS_DurabilityQosPolicy::writer_depth** (p. 1008) samples per instance in the DataWriter's queue if the **DDS_HistoryQosPolicy::depth** (p. 1164) is set to a higher value than **DDS_DurabilityQosPolicy::writer_depth** (p. 1008). Those additional samples past **DDS_DurabilityQosPolicy::writer_depth** (p. 1008) are not eligible to be sent in response to the TopicQuery.

A TopicQuery may select multiple samples at once. The writer will publish them periodically, independently from newly written samples. **DDS_TopicQueryDispatchQosPolicy::publication_period** (p. 1405) configures the frequency of that period and **DDS_TopicQueryDispatchQosPolicy::samples_per_period** (p. 1405) configures the maximum number of samples to publish each period.

If the DataWriter blocks during the publication of one of these samples, it will stop and try again the next period. (See **FooDataWriter::write** (p. 1951) for the conditions that may cause the write operation to block.)

All the DataWriters that belong to a single **DDSPublisher** (p. 1818) and enable TopicQueries share the same event thread, but each DataWriter schedules separate events. To configure that thread, see **DDS_AsynchronousPublisherQosPolicy::topic_query_publication_thread** (p. 802).

If the DataWriter is dispatching more than one TopicQuery at the same time, the configuration of this periodic event applies to all of them. For example, if a DataWriter receives two TopicQueries around the same time, the period is 1 second, the number of samples per period is 10, the first TopicQuery selects 5 samples, and the second one selects 8, the DataWriter will immediately attempt to publish all 5 for the first TopicQuery and 5 for the second one. After one second, it will publish the remaining 3 samples.

Entity:

DDSDataWriter (p. 1589)

Properties:

RxO (p. 556) = N/A
Changeable (p. 556) = **NO** (p. 556)

See also

DDSPublisher (p. 1818)

9.195.2 Member Data Documentation

9.195.2.1 enable

```
DDS_Boolean DDS_TopicQueryDispatchQosPolicy::enable
```

<<***extension***>> *(p. 213)* Allows this writer to dispatch TopicQueries

When set to true, this writer can receive and dispatch TopicQueries.

[default] DDS_BOOLEAN_FALSE (p. 513)

Categories

Topic Queries (p. 330), **Immutable** (p. 333), **Discovery Metadata** (p. 377)

9.195.2.2 publication_period

```
struct DDS_Duration_t DDS_TopicQueryDispatchQosPolicy::publication_period
```

<<***basic***>> *(p. 201)* <<***extension***>> *(p. 213)* Sets the periodic interval at which samples are published.

[default] 1 second

[range] [0,1 year]

Categories

Topic Queries (p. 330), **Immutable** (p. 333)

9.195.2.3 samples_per_period

```
DDS_Long DDS_TopicQueryDispatchQosPolicy::samples_per_period
```

<<***extension***>> *(p. 213)* Sets the maximum number of samples to publish in each publication_period.

[default] DDS_LENGTH_UNLIMITED (p. 636)

[range] [1, 100000000] or **DDS_LENGTH_UNLIMITED** (p. 636)

Categories

Topic Queries (p. 330), **Immutable** (p. 333)

9.196 DDS_TopicQuerySelection Struct Reference

<<***extension***>> *(p. 432)* Specifies the data query that defines a **DDSTopicQuery** (p. 1895)

Public Attributes

- char * **filter_class_name**

  The name of the filter to use.
- char * **filter_expression**

  The filter expression.
- struct **DDS_StringSeq filter_parameters**

  The query parameters.
- **DDS_TopicQuerySelectionKind kind**

  Indicates whether the sample selection is limited to cached samples or not.

9.196.1 Detailed Description

<<***extension***>> *(p. 432)* Specifies the data query that defines a **DDSTopicQuery** (p. 1895)

The query format is similar to the expression and parameters of a **DDSQueryCondition** (p. 1841) or a **DDSContentFilteredTopic** (p. 1551), as described in **DDSDomainParticipant::create_contentfilteredtopic_with_filter** (p. 1654).

There are two special queries:

- **DDS_TOPIC_QUERY_SELECTION_SELECT_ALL** (p. 160)
- **DDS_TOPIC_QUERY_SELECTION_USE_READER_CONTENT_FILTER** (p. 160)

See also

Queries and Filters Syntax (p. 184)

9.196.2 Member Data Documentation

9.196.2.1 filter_class_name

```
char* DDS_TopicQuerySelection::filter_class_name
```

The name of the filter to use.

Name of content filter to use. Must be one of the built-in filter names or previously registered with **DDSDomain↩ Participant::register_contentfilter** (p. 1632) on the same **DDSDomainParticipant** (p. 1619).

Built-in filter names are **DDS_SQLFILTER_NAME** (p. 61) and **DDS_STRINGMATCHFILTER_NAME** (p. 61).

[default] See **DDS_TOPIC_QUERY_SELECTION_USE_READER_CONTENT_FILTER** (p. 160)

9.196.2.2 filter_expression

```
char* DDS_TopicQuerySelection::filter_expression
```

The filter expression.

The expression to filter samples in the DataWriters. It follows the format described in **Queries and Filters Syntax** (p. 184).

[default] See **DDS_TOPIC_QUERY_SELECTION_USE_READER_CONTENT_FILTER** (p. 160)

See also

Queries and Filters Syntax (p. 184)

9.196.2.3 filter_parameters

```
struct DDS_StringSeq DDS_TopicQuerySelection::filter_parameters
```

The query parameters.

[default] See **DDS_TOPIC_QUERY_SELECTION_USE_READER_CONTENT_FILTER** (p. 160)

See also

DDSDomainParticipant::create_contentfilteredtopic_with_filter (p. 1654).

9.196.2.4 kind

```
DDS_TopicQuerySelectionKind DDS_TopicQuerySelection::kind
```

Indicates whether the sample selection is limited to cached samples or not.

[default] DDS_TOPIC_QUERY_SELECTION_KIND_HISTORY_SNAPSHOT (p. 160)

9.197 DDS_TransportBuiltinQosPolicy Struct Reference

*<<**extension**>> (p. 213)* Specifies which built-in transports are used.

Public Attributes

- **DDS_TransportBuiltinKindMask mask**

  *<<**basic**>> (p. 201) <<**extension**>> (p. 213) Specifies the built-in transports that are registered automatically when the **DDSDomainParticipant** (p. 1619) is enabled.*

9.197.1 Detailed Description

*<<**extension**>> (p. 213)* Specifies which built-in transports are used.

Three different transport plug-ins are built into the core RTI Connext libraries (for most supported target platforms): UDPv4, shared memory, and UDPv6.

This QoS policy allows you to control which of these built-in transport plug-ins are used by a **DDSDomainParticipant** (p. 1619). By default, only the UDPv4 and shared memory plug-ins are enabled (although on some embedded platforms, the shared memory plug-in is not available). In some cases, users will disable the shared memory transport when they do not want applications to use shared memory to communicate when running on the same node.

Entity:

DDSDomainParticipant (p. 1619)

Properties:

RxO (p. 556) = N/A
Changeable (p. 556) = **NO** (p. 556)

9.197.2 Member Data Documentation

9.197.2.1 mask

DDS_TransportBuiltinKindMask DDS_TransportBuiltinQosPolicy::mask

*<<**basic**>> (p. 201) <<**extension**>> (p. 213)* Specifies the built-in transports that are registered automatically when the **DDSDomainParticipant** (p. 1619) is enabled.

RTI Connext provides several built-in transports. Only those that are specified with this mask are registered automatically when the **DDSDomainParticipant** (p. 1619) is enabled.

[default] DDS_TRANSPORTBUILTIN_MASK_DEFAULT (p. 642)

Categories

Immutable (p. 333), **Transports** (p. 327)

9.198 DDS_TransportInfo_t Struct Reference

Contains the class_id and message_size_max of an installed transport.

Public Attributes

- **NDDS_Transport_ClassId_t class_id**

  The class_id identifies the transport associated with the message_size_max.
- **DDS_Long message_size_max**

  The maximum size of an RTPS message in bytes that can be sent or received by the transport plugin identified by the class_id.

9.198.1 Detailed Description

Contains the class_id and message_size_max of an installed transport.

9.198.2 Member Data Documentation

9.198.2.1 class_id

NDDS_Transport_ClassId_t DDS_TransportInfo_t::class_id

The class_id identifies the transport associated with the message_size_max.

9.198.2.2 message_size_max

DDS_Long DDS_TransportInfo_t::message_size_max

The maximum size of an RTPS message in bytes that can be sent or received by the transport plugin identified by the class_id.

9.199 DDS_TransportInfoSeq Struct Reference

Instantiates **FooSeq** (p. 1965) < **DDS_TransportInfo_t** (p. 1409) > .

9.199.1 Detailed Description

Instantiates **FooSeq** (p. 1965) < **DDS_TransportInfo_t** (p. 1409) > .

Instantiates:

<<***generic***>> *(p. 433)* **FooSeq** (p. 1965)

See also

DDS_TransportInfo_t (p. 1409)

9.200 DDS_TransportMulticastMapping_t Struct Reference

<<***extension***>> *(p. 213)* Type representing a list of multicast mapping elements.

Public Attributes

- char * **addresses**

  <<***extension***>> *(p. 213) A string containing a comma-separated list of IP addresses or IP address ranges to be used to receive multicast traffic for the entity with a topic that matches the* ***DDS_TransportMulticastMapping_t::topic_expression*** *(p. 1411).*

- char * **topic_expression**

  <<***extension***>> *(p. 213) A regular expression that will be used to map topic names to corresponding multicast receive addresses.*

- struct **DDS_TransportMulticastMappingFunction_t** **mapping_function**

  <<***extension***>> *(p. 213) Specifies a function that will define the mapping between a topic name and a specific multicast address from a list of addresses.*

9.200.1 Detailed Description

<<***extension***>> *(*p. *213)* Type representing a list of multicast mapping elements.

A multicast mapping element specifies a string containing a list of IP addresses, a topic expression and a mapping function.

QoS:

DDS_TransportMulticastMappingQosPolicy (p. 1413)

9.200.2 Member Data Documentation

9.200.2.1 addresses

`char* DDS_TransportMulticastMapping_t::addresses`

<<***extension***>> *(*p. *213)* A string containing a comma-separated list of IP addresses or IP address ranges to be used to receive *multicast* traffic for the entity with a topic that matches the **DDS_TransportMulticastMapping_t::topic_↩ expression** (p. 1411).

The string must contain IPv4 or IPv6 addresses separated by commas. For example: "239.255.100.1,239.255.100.↩ 2,239.255.100.3"

You may specify ranges of addresses by enclosing the start address and the end address in square brackets. For example: "[239.255.100.1,239.255.100.3]"

You may combine the two approaches. For example:

"239.255.200.1,[239.255.100.1,239.255.100.3], 239.255.200.3"

IPv4 addresses must be specified in Dot-decimal notation.

IPv6 addresses must be specified using 8 groups of 16-bit hexadecimal values separated by colons. For example: FF00:0000:0000:0000:0202:B3FF:FE1E:8329

Leading zeroes can be skipped. For example: "FF00:0:0:0:202:B3FF:FE1E:8329"

You may replace a consecutive number of zeroes with a double colon, but only once within an address. For example: "FF00::202:B3FF:FE1E:8329"

[default] NULL

9.200.2.2 topic_expression

```
char* DDS_TransportMulticastMapping_t::topic_expression
```

<<***extension***>> *(*p. *213)* A regular expression that will be used to map topic names to corresponding multicast receive addresses.

A topic name must match the expression before a corresponding address is assigned.

[default] NULL

9.200.2.3 mapping_function

```
struct DDS_TransportMulticastMappingFunction_t DDS_TransportMulticastMapping_t::mapping_function
```

<<***extension***>> *(*p. *213)* Specifies a function that will define the mapping between a topic name and a specific multicast address from a list of addresses.

This function is optional. If not specified, the middleware will use a hash function to perform the mapping.

9.201 DDS_TransportMulticastMappingFunction_t Struct Reference

Type representing an external mapping function.

Public Attributes

- char * **dll**

  Specifies a dynamic library that contains a mapping function.
- char * **function_name**

  Specifies the name of a mapping function.

9.201.1 Detailed Description

Type representing an external mapping function.

A mapping function is defined by a dynamic library name and a function name.

QoS:

DDS_TransportMulticastMappingQosPolicy (p. 1413)

9.201.2 Member Data Documentation

9.201.2.1 dll

```
char* DDS_TransportMulticastMappingFunction_t::dll
```

Specifies a dynamic library that contains a mapping function.

A relative or absolute path can be specified.

If the name is specified as "foo", the library name on Linux systems will be libfoo.so; on Windows systems it will be foo.dll.

[default] NULL

9.201.2.2 function_name

```
char* DDS_TransportMulticastMappingFunction_t::function_name
```

Specifies the name of a mapping function.

This function must be implemented in the library specified in **DDS_TransportMulticastMappingFunction_t::dll** (p. 1412).

The function must implement the following interface:

int function(const char* topic_name, int numberOfAddresses);

The function must return an integer that indicates the *index* of the address to use for the given `topic_name`. For example, if the first address in the list should be used, it must return 0; if the second address in the list should be used, it must return 1, etc.

9.202 DDS_TransportMulticastMappingQosPolicy Struct Reference

<<***extension***>> *(p. 213)* Specifies a list of topic_expressions and multicast addresses that can be used by an Entity with a specific topic name to receive data.

Public Attributes

- struct **DDS_TransportMulticastMappingSeq value**

  <<***extension***>> *(p. 213) A sequence of multicast communication mappings.*

9.202.1 Detailed Description

<<***extension***>> *(p. 213)* Specifies a list of topic_expressions and multicast addresses that can be used by an Entity with a specific topic name to receive data.

This QoS policy provides an alternate way to assign multicast receive addresses to DataReaders. It allows you to perform multicast configuration at the **DDSDomainParticipant** (p. 1619) level.

To use multicast communication *without* this QoS policy, you must explicitly assign a multicast receive address on each **DDSDataReader** (p. 1556). This can quickly become difficult to configure as more DataReaders of different topics and multicast addresses are added.

With this QoS policy, you can configure a set of multicast addresses on the **DDSDomainParticipant** (p. 1619); those addresses will then be automatically assigned to the DomainParticipant's DataReaders. A single configuration on the **DDSDomainParticipant** (p. 1619) can thus replace per-DataReader configuration.

On the DomainParticipant, the set of assignable addresses can be configured for specific topics. Addresses are configured on topics because efficient usage of multicast will have all DataWriters and DataReaders of a single topic using the same multicast address.

You can specify a mapping between a topic's name and a multicast address. For example, topic 'A' can be assigned to address 239.255.1.1 and topic 'B' can be assigned to address 239.255.1.2.

You can use filter expressions to configure a subset of topics to use a specific list of addresses. For example, suppose topics "X", "Y" and "Z" need to be sent to any address within the range [239.255.1.1, 239.255.1.255]. You can specify an expression on the topic name (e.g. "[X-Z]") corresponding to that range of addresses. Then the DomainParticipant will select an address for a topic whose name matches the expression.

The middleware will use a hash function to perform the mapping from topic to address. Alternatively, you can specify a pluggable mapping function.

IMPORTANT: All the strings defined in each element of the sequence must be assigned using RTI_String_dup("foo");. For example:

mcastMappingElement->addresses = DDS_String_dup("[239.255.1.1,239.255.1.255]");

NOTE: To use this QoS policy, you must:

- Set **DDS_TransportMulticastQosPolicy::kind** (p. 1416) to **DDS_AUTOMATIC_TRANSPORT_MULTICAST_QOS** (p. 646).
- Set the first element in **DDSDataReader** (p. 1556)'s **DDS_TransportMulticastQosPolicy::value** (p. 1416) with an empty address.

Entity:

DDSDomainParticipant (p. 1619)

Properties:

RxO (p. 556) = N/A
Changeable (p. 556) = **NO** (p. 556)

9.202.2 Member Data Documentation

9.202.2.1 value

`struct` **`DDS_TransportMulticastMappingSeq`** `DDS_TransportMulticastMappingQosPolicy::value`

<<***extension***>> *(*p. *213)* A sequence of multicast communication mappings.

[default] Empty sequence.

Categories

Immutable (p. 333), **Transports** (p. 327), **Data Availability** (p. 261)

9.203 DDS_TransportMulticastMappingSeq Struct Reference

Declares IDL `sequence<` **DDS_TransportMulticastMapping_t** (p. 1410) `>`

9.203.1 Detailed Description

Declares IDL `sequence<` **DDS_TransportMulticastMapping_t** (p. 1410) `>`

Instantiates:

<<***generic***>> *(*p. *433)* **FooSeq** (p. 1965)

See also

DDS_TransportMulticastMapping_t (p. 1410)

9.204 DDS_TransportMulticastQosPolicy Struct Reference

<<***extension***>> *(*p. *213)* Specifies the multicast address on which a **DDSDataReader** (p. 1556) wants to receive its data. It can also specify a port number as well as a subset of the available (at the **DDSDomainParticipant** (p. 1619) level) transports with which to receive the multicast data.

Public Attributes

- struct **DDS_TransportMulticastSettingsSeq value**

  <<***basic***>> *(*p. *201)* <<***extension***>> (p. *213) A sequence of multicast communications settings.*
- **DDS_TransportMulticastQosPolicyKind kind**

  <<***extension***>> *(*p. *213) A value that specifies a way to determine how to obtain the multicast address.*

9.204.1 Detailed Description

<<***extension***>> *(p. 213)* Specifies the multicast address on which a **DDSDataReader** (p. 1556) wants to receive its data. It can also specify a port number as well as a subset of the available (at the **DDSDomainParticipant** (p. 1619) level) transports with which to receive the multicast data.

By default, a **DDSDataWriter** (p. 1589) will send individually addressed packets for each **DDSDataReader** (p. 1556) that subscribes to the topic of the DataWriter – this is known as unicast delivery. Thus, as many copies of the data will be sent over the network as there are DataReaders for the data. The network bandwidth used by a DataWriter will thus increase linearly with the number of DataReaders.

Multicast addressing (on UDP/IP transports) allows multiple DataReaders to receive the *same* network packet. By using multicast, a **DDSDataWriter** (p. 1589) can send a single network packet that is received by all subscribing applications. Thus the network bandwidth usage will be constant, independent of the number of DataReaders.

Coordinating the multicast address specified by DataReaders can help optimize network bandwidth usage in systems where there are multiple DataReaders for the same **DDSTopic** (p. 1885).

Entity:

DDSDataReader (p. 1556)

Properties:

RxO (p. 556) = N/A
Changeable (p. 556) = **NO** (p. 556)

9.204.2 Member Data Documentation

9.204.2.1 value

`struct` **`DDS_TransportMulticastSettingsSeq`** `DDS_TransportMulticastQosPolicy::value`

<<***basic***>> *(p. 201)* <<***extension***>> *(p. 213)* A sequence of multicast communications settings.

An empty sequence means that multicast is not used by the entity.

The RTPS wire protocol currently limits the maximum number of multicast locators to

1. Note that this is a hard limit that cannot be increased. However, this limit can be decreased by configuring the 'dds.domain_participant.max_announced_locator_list_size' property in the **DDS_PropertyQosPolicy** (p. 1260) associated with the **DDS_DomainParticipantQos** (p. 971).

[default] Empty sequence.

Categories

Immutable (p. 333), **Transports** (p. 327), **Discovery Metadata** (p. 377)

9.204.2.2 kind

DDS_TransportMulticastQosPolicyKind DDS_TransportMulticastQosPolicy::kind

<<***extension***>> *(p. 213)* A value that specifies a way to determine how to obtain the multicast address.

This field can be set to one of the following two values: **DDS_AUTOMATIC_TRANSPORT_MULTICAST_QOS** (p. 646) or **DDS_UNICAST_ONLY_TRANSPORT_MULTICAST_QOS** (p. 646).

- If it is set to **DDS_AUTOMATIC_TRANSPORT_MULTICAST_QOS** (p. 646), the behavior will depend on the **DDS_TransportMulticastQosPolicy::value** (p. 1416):
  - If **DDS_TransportMulticastQosPolicy::value** (p. 1416) does not have any elements, then multicast will not be used.
  - If **DDS_TransportMulticastQosPolicy::value** (p. 1416) first element has an empty address, then the address will be obtained from **DDS_TransportMulticastMappingQosPolicy** (p. 1413).
  - If none of the elements in **DDS_TransportMulticastQosPolicy::value** (p. 1416) is empty, and at least one element has a valid address, then that address will be used.
- If it is set to **DDS_UNICAST_ONLY_TRANSPORT_MULTICAST_QOS** (p. 646), then multicast will not be used.

[default] DDS_AUTOMATIC_TRANSPORT_MULTICAST_QOS (p. 646)

Categories

Immutable (p. 333), **Transports** (p. 327), **Data Availability** (p. 261)

9.205 DDS_TransportMulticastSettings_t Struct Reference

Type representing a list of multicast locators.

Public Attributes

- struct **DDS_StringSeq transports**

  <<***extension***>> *(p. 213) A sequence of transport aliases that specifies the transports on which to receive multicast traffic for the entity.*
- char * **receive_address**

  <<***extension***>> *(p. 213) The multicast group address on which the entity can receive data.*
- **DDS_Long receive_port**

  <<***extension***>> *(p. 213) The multicast port on which the entity can receive data.*

9.205.1 Detailed Description

Type representing a list of multicast locators.

A multicast locator specifies a transport class, a multicast address, and a multicast port number on which messages can be received by an entity.

QoS:

DDS_TransportMulticastQosPolicy (p. 1415)

9.205.2 Member Data Documentation

9.205.2.1 transports

```
struct  DDS_StringSeq DDS_TransportMulticastSettings_t::transports
```

<<***extension***>> *(p. 213)* A sequence of transport aliases that specifies the transports on which to receive *multicast* traffic for the entity.

Of the transport instances available to the entity, only those with aliases matching an alias in this sequence are used to subscribe to the multicast group addresses. Thus, this list of aliases sub-selects from the transport s available to the entity.

An empty sequence is a special value that specifies all the transports available to the entity.

The memory for the strings in this sequence is managed according to the conventions described in **String Conventions** (p. 758). In particular, be careful to avoid a situation in which RTI Connext allocates a string on your behalf and you then reuse that string in such a way that RTI Connext believes it to have more memory allocated to it than it actually does.

Alias names for the builtin transports are defined in **TRANSPORT_BUILTIN** (p. 641).

[default] Empty sequence; i.e. all the transports available to the entity.

[range] Any sequence of non-null, non-empty strings.

9.205.2.2 receive_address

```
char* DDS_TransportMulticastSettings_t::receive_address
```

<<***extension***>> *(p. 213)* The multicast group address on which the entity can receive data.

Must must be an address in the proper format (see **Address Format** (p. 664)).

[default] NONE/INVALID. Required to specify a multicast group address to join.

[range] A valid IPv4 or IPv6 multicast address.

See also

Address Format (p. 664)

9.205.2.3 receive_port

```
DDS_Long DDS_TransportMulticastSettings_t::receive_port
```

<<***extension***>> *(*p. *213)* The multicast port on which the entity can receive data.

[default] 0, which implies that the actual port number is determined by a formula as a function of the `domain_id` (see **DDS_WireProtocolQosPolicy::participant_id** (p. 1514)).

[range] [0,0xffffffff]

9.206 DDS_TransportMulticastSettingsSeq Struct Reference

Declares IDL `sequence<` **DDS_TransportMulticastSettings_t** (p. 1417) `>`

9.206.1 Detailed Description

Declares IDL `sequence<` **DDS_TransportMulticastSettings_t** (p. 1417) `>`

Instantiates:

<<***generic***>> *(*p. *433)* **FooSeq** (p. 1965)

See also

DDS_TransportMulticastSettings_t (p. 1417)

9.207 DDS_TransportPriorityQosPolicy Struct Reference

<<***spec***>> *(*p. *209)* This QoS policy allows the application to take advantage of transports that are capable of sending messages with different priorities.

Public Attributes

- **DDS_Long value**

  <<***spec***>> (p. *209) This policy is a hint to the infrastructure as to how to set the priority of the underlying transport used to send the data.*

9.207.1 Detailed Description

<<***spec***>> *(p. 209)* This QoS policy allows the application to take advantage of transports that are capable of sending messages with different priorities.

The Transport Priority QoS policy is optional and only supported on certain OSs and transports. It allows you to specify on a per-DataWriter or DataReader basis that the data sent by that endpoint is of a different priority.

The DDS specification does not indicate how a DDS implementation should treat data of different priorities. It is often difficult or impossible for DDS implementations to treat data of higher priority differently than data of lower priority, especially when data is being sent (delivered to a physical transport) directly by the thread that called **FooDataWriter::write** (p. 1951). Also, many physical network transports themselves do not have a end-user controllable level of data packet priority.

Entity:

DDSDataWriter (p. 1589), **DDSDataReader** (p. 1556), **DDSTopic** (p. 1885)

Properties:

RxO (p. 556) = N/A
Changeable (p. 556) = **NO** (p. 556)

9.207.2 Usage

In RTI Connext, for the IP-based transports (UDPv4, UDPv6, Real-Time WAN, and TCP), the value set in the Transport Priority QoS policy can be used to set the differentiated services field (DS field) bits of the IPv4 and IPv6 headers for datagrams sent by a **DDSDataWriter** (p. 1589) or **DDSDataReader** (p. 1556). It is platform-dependent on how and whether setting the DS field has an effect. Some platforms may require external permissions in order to set the DS field.

The transport priority value is not provided as is to the transports, but transformed according to the following fields: transport_priority_mask (for example, see **NDDS_Transport_UDPv4_Property_t::transport_priority_mask** (p. 2062)), transport_priority_mapping_low (for example, see **NDDS_Transport_UDPv4_Property_t::transport_priority_mapping_low** (p. 2063)), and transport_priority_mapping_high (for example, see **NDDS_Transport_UDPv4_Property_t::transport_priority_mapping_high** (p. 2063)). If you want the priority value to be exactly equal to the DS value, then the only change you need to make is to set transport_priority_mask to `0xff` (and keep the transport_priority_mapping_low and transport_priority_mapping_high defaults).

It is incorrect to assume that using the Transport Priority QoS policy will have any effect at all on the end-to-end delivery of data from a **DDSDataWriter** (p. 1589) and a **DDSDataReader** (p. 1556). All network elements, including switches and routers, must have the capability and be enabled to actually use the DS field to treat higher priority packets differently. Thus the ability to use the Transport Priority QoS policy must be designed and configured at a *system* level; just turning it on in an application may have no effect at all at a transport level.

For additional details on how to set the DS field in IP-based transports, see the "TRANSPORT_PRIORITY QosPolicy" section of the `User's Manual`.

9.207.3 Member Data Documentation

9.207.3.1 value

```
DDS_Long DDS_TransportPriorityQosPolicy::value
```

<<***spec***>> *(p. 209)* This policy is a hint to the infrastructure as to how to set the priority of the underlying transport used to send the data.

You may choose any value within the range of a 32-bit signed integer; higher values indicate higher priority. However, any further interpretation of this policy is specific to a particular transport and a particular DDS implementation. For example, a particular transport is permitted to treat a range of priority values as equivalent to one another.

[default] 0

Categories

Immutable (p. 333), **Transports** (p. 327), **Concurrency** (p. 258)

9.208 DDS_TransportSelectionQosPolicy Struct Reference

<<***extension***>> *(p. 213)* Specifies the physical transports a **DDSDataWriter** (p. 1589) or **DDSDataReader** (p. 1556) may use to send or receive data.

Public Attributes

- struct **DDS_StringSeq enabled_transports**

  <<***extension***>> *(p. 213) A sequence of transport aliases that specifies the transport instances available for use by the entity.*

9.208.1 Detailed Description

<<***extension***>> *(p. 213)* Specifies the physical transports a **DDSDataWriter** (p. 1589) or **DDSDataReader** (p. 1556) may use to send or receive data.

An application may be simultaneously connected to many different physical transports, e.g., Ethernet, Infiniband, shared memory, VME backplane, and wireless. By default, RTI Connext will use up to 16 transports to deliver data from a DataWriter to a DataReader.

This QoS policy can be used to both limit and control which of the application's available transports may be used by a **DDSDataWriter** (p. 1589) to send data or by a **DDSDataReader** (p. 1556) to receive data.

Entity:

DDSDataReader (p. 1556), **DDSDataWriter** (p. 1589)

Properties:

RxO (p. 556) = N/A
Changeable (p. 556) = **NO** (p. 556)

9.208.2 Member Data Documentation

9.208.2.1 enabled_transports

```
struct DDS_StringSeq DDS_TransportSelectionQosPolicy::enabled_transports
```

<<***extension***>> *(p. 213)* A sequence of transport aliases that specifies the transport instances available for use by the entity.

Of the transport instances installed with the **DDSDomainParticipant** (p. 1619), only those with aliases matching an alias in this sequence are available to the entity.

Thus, this list of aliases sub-selects from the transports available to the **DDSDomainParticipant** (p. 1619).

An empty sequence is a special value that specifies all the transports installed with the **DDSDomainParticipant** (p. 1619).

The memory for the strings in this sequence is managed according to the conventions described in **String Conventions** (p. 758). In particular, be careful to avoid a situation in which RTI Connext allocates a string on your behalf and you then reuse that string in such a way that RTI Connext believes it to have more memory allocated to it than it actually does.

Alias names for the builtin transports are defined in **TRANSPORT_BUILTIN** (p. 641). These alias names are case sensitive and should be written in lowercase.

[default] Empty sequence; i.e. all the transports installed with and available to the **DDSDomainParticipant** (p. 1619).

[range] A sequence of non-null, non-empty strings.

See also

DDS_DomainParticipantQos::transport_builtin (p. 974).

Categories

Immutable (p. 333), **Transports** (p. 327)

9.209 DDS_TransportUnicastQosPolicy Struct Reference

<<***extension***>> *(p. 213)* Specifies a subset of transports and a port number that can be used by an Entity to receive data.

Public Attributes

- struct **DDS_TransportUnicastSettingsSeq value**

  <<***extension***>> *(p. 213) A sequence of unicast communication settings.*

9.209.1 Detailed Description

<<***extension***>> *(p. 213)* Specifies a subset of transports and a port number that can be used by an Entity to receive data.

Entity:

DDSDomainParticipant (p. 1619), **DDSDataReader** (p. 1556), **DDSDataWriter** (p. 1589)

Properties:

RxO (p. 556) = N/A
Changeable (p. 556) = **NO** (p. 556)

9.209.2 Usage

RTI Connext may send data to a variety of Entities, not just DataReaders. For example, reliable DataWriters may receive ACK/NACK packets from reliable DataReaders.

During discovery, each **DDSEntity** (p. 1731) announces to remote applications a list of (up to 16) unicast addresses to which the remote application should send data (either user data packets or reliable protocol meta-data such as ACK/NACKs and heartbeats). Note that this is a hard limit that cannot be increased. However, this limit can be decreased by configuring the 'dds.domain_participant.max_announced_locator_list_size' property in the **DDS_PropertyQosPolicy** (p. 1260) associated with the **DDS_DomainParticipantQos** (p. 971).

Note: RTI Connext internally uses several dynamic locator lists to keep track of configured unicast locators. The initial size of these lists can be configured with the **DDSDomainParticipant** (p. 1619) property 'dds.domain_participant.initial_locator_list_size'. By default, the initial size is derived from local information, such as the number of local locators. To prevent dynamic resizing at runtime, set the property's value to the maximum number of locators that a single participant or endpoint may advertise at any given time in your system.

By default, the list of addresses is populated automatically with values obtained from the enabled transport plug-ins allowed to be used by the Entity (see **DDS_TransportBuiltinQosPolicy** (p. 1408) and **DDS_TransportSelectionQosPolicy** (p. 1421)). Also, the associated ports are automatically determined (see **DDS_RtpsWellKnownPorts_t** (p. 1338)).

Use this QoS policy to manually set the receive address list for an Entity. You may optionally set a port to use a non-default receive port as well. Only the first 16 addresses will be used.

RTI Connext will create a receive thread for every unique port number that it encounters (on a per transport basis).

- For a **DDSDomainParticipant** (p. 1619), this QoS policy sets the default list of addresses used by other applications to send user data for local DataReaders.

- For a **DDSDataReader** (p. 1556), if set, then other applications will use the specified list of addresses to send user data (and reliable protocol packets for reliable DataReaders). Otherwise, if not set, the other applications will use the addresses set by the **DDSDomainParticipant** (p. 1619).

- For a reliable **DDSDataWriter** (p. 1589), if set, then other applications will use the specified list of addresses to send reliable protocol packets (ACKS/NACKS) on the behalf of reliable DataReaders. Otherwise, if not set, the other applications will use the addresses set by the **DDSDomainParticipant** (p. 1619).

9.209.3 Member Data Documentation

9.209.3.1 value

`struct` **DDS_TransportUnicastSettingsSeq** `DDS_TransportUnicastQosPolicy::value`

<<***extension***>> *(p. 213)* A sequence of unicast communication settings.

An empty sequence means that applicable defaults specified by elsewhere (e.g. **DDS_DomainParticipantQos**↩ **::default_unicast** (p. 974)) should be used.

The RTPS wire protocol currently limits the maximum number of unicast locators to

1. Note that this is a hard limit that cannot be increased. However, this limit can be decreased by configuring the 'dds.domain_participant.max_announced_locator_list_size' property in the **DDS_PropertyQosPolicy** (p. 1260) associated with the **DDS_DomainParticipantQos** (p. 971).

[default] Empty sequence.

See also

DDS_DomainParticipantQos::default_unicast (p. 974)

Categories

Immutable (p. 333), **Transports** (p. 327), **Discovery Metadata** (p. 377)

9.210 DDS_TransportUnicastSettings_t Struct Reference

Type representing a list of unicast locators.

Public Attributes

- struct **DDS_StringSeq transports**

  <<***extension***>> *(p. 213) A sequence of transport aliases that specifies the unicast interfaces on which to receive unicast traffic for the entity.*
- **DDS_Long receive_port**

  <<***extension***>> *(p. 213) The unicast port on which the entity can receive data.*

9.210.1 Detailed Description

Type representing a list of unicast locators.

A unicast locator specifies a transport class, a unicast address, and a unicast port number on which messages can be received by an entity.

QoS:

DDS_TransportUnicastQosPolicy (p. 1422)

9.210.2 Member Data Documentation

9.210.2.1 transports

`struct` **DDS_StringSeq** DDS_TransportUnicastSettings_t::transports

<<***extension***>> *(p. 213)* A sequence of transport aliases that specifies the unicast interfaces on which to receive *unicast* traffic for the entity.

Of the transport instances available to the entity, only those with aliases matching an alias on this sequence are used to determine the unicast interfaces used by the entity.

Thus, this list of aliases sub-selects from the transports available to the entity.

Each unicast interface on a transport results in a unicast locator for the entity.

An empty sequence is a special value that specifies all the transports available to the entity.

The memory for the strings in this sequence is managed according to the conventions described in **String Conventions** (p. 758). In particular, be careful to avoid a situation in which RTI Connext allocates a string on your behalf and you then reuse that string in such a way that RTI Connext believes it to have more memory allocated to it than it actually does.

Alias names for the builtin transports are defined in **TRANSPORT_BUILTIN** (p. 641).

[default] Empty sequence; i.e. all the transports available to the entity.
[range] Any sequence of non-null, non-empty strings.

9.210.2.2 receive_port

DDS_Long DDS_TransportUnicastSettings_t::receive_port

<<***extension***>> *(p. 213)* The unicast port on which the entity can receive data.

Must be an *unused* unicast port on the system.

[default] 0, which implies that the actual port number is determined by a formula as a function of the `domain_id`, and the **DDS_WireProtocolQosPolicy::participant_id** (p. 1514).

[range] [0,0xffffffff]

See also

DDS_WireProtocolQosPolicy::participant_id (p. 1514).

9.211 DDS_TransportUnicastSettingsSeq Struct Reference

Declares IDL `sequence<` **DDS_TransportUnicastSettings_t** (p. 1424) `>`

9.211.1 Detailed Description

Declares IDL `sequence<` **DDS_TransportUnicastSettings_t** (p. 1424) `>`

Instantiates:

*<<**generic**>> (p. 433)* **FooSeq** (p. 1965)

See also

DDS_TransportUnicastSettings_t (p. 1424)

9.212 DDS_TrustAlgorithmRequirements Struct Reference

Type to describe Trust Plugins algorithm requirements for an entity.

Public Attributes

- **DDS_TrustAlgorithmSet supported_mask**
  Trust Plugins algorithms that an entity supports.
- **DDS_TrustAlgorithmSet required_mask**
  Trust Plugins algorithms that an entity uses.

9.212.1 Detailed Description

Type to describe Trust Plugins algorithm requirements for an entity.

9.212.2 Member Data Documentation

9.212.2.1 supported_mask

DDS_TrustAlgorithmSet DDS_TrustAlgorithmRequirements::supported_mask

Trust Plugins algorithms that an entity supports.

9.212.2.2 required_mask

DDS_TrustAlgorithmSet DDS_TrustAlgorithmRequirements::required_mask

Trust Plugins algorithms that an entity uses.

9.213 DDS_TypeAllocationParams_t Struct Reference

Configures whether or not to allocate pointer and optional members.

Public Member Functions

- struct **DDS_TypeAllocationParams_t** & **set_allocate_pointers** (**DDS_Boolean** allocate)
- struct **DDS_TypeAllocationParams_t** & **set_allocate_optional_members** (**DDS_Boolean** allocate)

Public Attributes

- **DDS_Boolean allocate_pointers**

  Whether to allocate pointer members or not.
- **DDS_Boolean allocate_optional_members**

  Whether to allocate optional members or not.

9.213.1 Detailed Description

Configures whether or not to allocate pointer and optional members.

By default pointers are allocated and optional members are not.

This structure is also defined in xcdr.1.0/srcC/typeObject/TypeObjectInfrastructure.h. If changes are made, please ensure they are made in both locations.

Examples

HelloWorld.cxx, and **HelloWorldPlugin.cxx**.

9.213.2 Member Function Documentation

9.213.2.1 set_allocate_pointers()

```
struct DDS_TypeAllocationParams_t & DDS_TypeAllocationParams_t::set_allocate_pointers (
            DDS_Boolean allocate ) [inline]
```

Whether to allocate pointer members or not

9.213.2.2 set_allocate_optional_members()

```
struct DDS_TypeAllocationParams_t & DDS_TypeAllocationParams_t::set_allocate_optional_members (
            DDS_Boolean allocate ) [inline]
```

Whether to allocate optional members or not

9.213.3 Member Data Documentation

9.213.3.1 allocate_pointers

```
DDS_Boolean DDS_TypeAllocationParams_t::allocate_pointers
```

Whether to allocate pointer members or not.

Examples

HelloWorld.cxx, and **HelloWorldPlugin.cxx**.

9.213.3.2 allocate_optional_members

```
DDS_Boolean DDS_TypeAllocationParams_t::allocate_optional_members
```

Whether to allocate optional members or not.

9.214 DDS_TypeCode Struct Reference

The definition of a particular data type, which you can use to inspect the name, members, and other properties of types generated with rtiddsgen (see the `Code Generator User's Manual`) or to modify types you define yourself at runtime.

Public Member Functions

- **DDS_TCKind kind** (**DDS_ExceptionCode_t** &ex) const

  *Gets the **DDS_TCKind** (p. 89) value of a type code.*
- **DDS_ExtensibilityKind extensibility_kind** (**DDS_ExceptionCode_t** &ex) const

  *Gets the **DDS_ExtensibilityKind** (p. 89) value of a type code.*
- **DDS_Boolean equal** (const **DDS_TypeCode** *tc, **DDS_ExceptionCode_t** &ex) const

  *Compares two **DDS_TypeCode** (p. 1428) objects for equality.*
- **DDS_Boolean is_assignable_from** (const **DDS_TypeCode** *from, **DDS_ExceptionCode_t** &ex) const

  Check if this type is assignable from another type.
- const char * **name** (**DDS_ExceptionCode_t** &ex) const

  *Retrieves the simple name identifying this **DDS_TypeCode** (p. 1428) object within its enclosing scope.*
- **DDS_UnsignedLong member_count** (**DDS_ExceptionCode_t** &ex) const

  Returns the number of members of the type code.
- const char * **member_name** (**DDS_UnsignedLong** index, **DDS_ExceptionCode_t** &ex) const

  Returns the name of a type code member identified by the given index.
- **DDS_UnsignedLong find_member_by_name** (const char * **name**, **DDS_ExceptionCode_t** &ex) const

  Get the index of the member of the given name.
- **DDS_TypeCode** * **member_type** (**DDS_UnsignedLong** index, **DDS_ExceptionCode_t** &ex) const

  *Retrieves the **DDS_TypeCode** (p. 1428) object describing the type of the member identified by the given index.*
- **DDS_UnsignedLong member_label_count** (**DDS_UnsignedLong** index, **DDS_ExceptionCode_t** &ex) const

  Returns the number of labels associated to the index-th union member.
- **DDS_Long member_label** (**DDS_UnsignedLong** member_index, **DDS_UnsignedLong** label_index, **DDS_ExceptionCode_t** &ex) const

  Return the label_index-th label associated to the member_index-th member.
- **DDS_Long member_ordinal** (**DDS_UnsignedLong** index, **DDS_ExceptionCode_t** &ex) const

  Returns the ordinal that corresponds to the index-th enum value.
- **DDS_Boolean is_member_key** (**DDS_UnsignedLong** index, **DDS_ExceptionCode_t** &ex) const

  Function that tells if a member is a key or not.
- **DDS_Boolean is_member_required** (**DDS_UnsignedLong** index, **DDS_ExceptionCode_t** &ex) const

  Indicates whether a given member of a type is required to be present in every sample of that type.
- **DDS_Boolean is_member_pointer** (**DDS_UnsignedLong** index, **DDS_ExceptionCode_t** &ex) const

  Function that tells if a member is a pointer or not.
- **DDS_Boolean is_member_bitfield** (**DDS_UnsignedLong** index, **DDS_ExceptionCode_t** &ex) const

  Function that tells if a member is a bitfield or not.
- **DDS_Short member_bitfield_bits** (**DDS_UnsignedLong** index, **DDS_ExceptionCode_t** &ex) const

  Returns the number of bits of a bitfield member.
- **DDS_Visibility member_visibility** (**DDS_UnsignedLong** index, **DDS_ExceptionCode_t** &ex) const

  Returns the constant that indicates the visibility of the index-th member.
- **DDS_TypeCode** * **discriminator_type** (**DDS_ExceptionCode_t** &ex) const

  Returns the discriminator type code.
- **DDS_UnsignedLong length** (**DDS_ExceptionCode_t** &ex) const

  Returns the number of elements in the type described by this type code.
- **DDS_UnsignedLong array_dimension_count** (**DDS_ExceptionCode_t** &ex) const

  This function returns the number of dimensions of an array type code.
- **DDS_UnsignedLong array_dimension** (**DDS_UnsignedLong** index, **DDS_ExceptionCode_t** &ex) const

This function returns the index-th dimension of an array type code.

- **DDS_UnsignedLong element_count** (**DDS_ExceptionCode_t** &ex) const

  The number of elements in an array.

- **DDS_TypeCode** * **content_type** (**DDS_ExceptionCode_t** &ex) const

  *Returns the **DDS_TypeCode** (p. 1428) object representing the type for the members of the object described by this **DDS_TypeCode** (p. 1428) object.*

- **DDS_Boolean is_alias_pointer** (**DDS_ExceptionCode_t** &ex) const

  Function that tells if an alias is a pointer or not.

- **DDS_Long default_index** (**DDS_ExceptionCode_t** &ex) const

  Returns the index of the default member, or -1 if there is no default member.

- **DDS_TypeCode** * **concrete_base_type** (**DDS_ExceptionCode_t** &ex) const

  *Returns the **DDS_TypeCode** (p. 1428) that describes the concrete base type of the value type that this **DDS_TypeCode** (p. 1428) object describes.*

- **DDS_ValueModifier type_modifier** (**DDS_ExceptionCode_t** &ex) const

  *Returns a constant indicating the modifier of the value type that this **DDS_TypeCode** (p. 1428) object describes.*

- **DDS_Long member_id** (**DDS_UnsignedLong** index, **DDS_ExceptionCode_t** &ex) const

  Returns the ID of the TypeCode member identified by the given index.

- **DDS_UnsignedLong find_member_by_id** (**DDS_Long** id, **DDS_ExceptionCode_t** &ex) const

  Get the index of the member of the given ID.

- **DDS_UnsignedLong get_type_object_serialized_size** (**DDS_ExceptionCode_t** &ex) const

  *Gets the serialized size of the TypeObject v1 created from this **DDS_TypeCode** (p. 1428).*

- **DDS_UnsignedLong get_cdr_serialized_sample_max_size** (**DDS_DataRepresentationId_t** representation_id, **DDS_ExceptionCode_t** &ex) const

  ***[DEPRECATED]** Gets the maximum serialized size of samples of this type*

- **DDS_UnsignedLong cdr_serialized_sample_max_size** (**DDS_DataRepresentationId_t** representation_id, **DDS_ExceptionCode_t** &ex) const

  Gets the maximum serialized size of samples of this type.

- **DDS_UnsignedLong cdr_serialized_sample_min_size** (**DDS_DataRepresentationId_t** representation_id, **DDS_ExceptionCode_t** &ex) const

  Gets the minimum serialized size of samples of this type.

- **DDS_UnsignedLong cdr_serialized_sample_key_max_size** (**DDS_DataRepresentationId_t** representation_id, **DDS_ExceptionCode_t** &ex) const

  Gets the maximum serialized size of sample keys of this type.

- **DDS_UnsignedLong cdr_serialized_sample_max_size** (**DDS_ExceptionCode_t** &ex) const

  Gets the maximum serialized size of samples of this type.

- **DDS_UnsignedLong cdr_serialized_sample_min_size** (**DDS_ExceptionCode_t** &ex) const

  Gets the minimum serialized size of samples of this type.

- **DDS_UnsignedLong cdr_serialized_sample_key_max_size** (**DDS_ExceptionCode_t** &ex) const

  Gets the maximum serialized size of sample keys of this type.

- **DDS_UnsignedLong add_member_to_enum** (const char * **name**, **DDS_Long** ordinal, **DDS_ExceptionCode_t** &ex)

  *Add a new enumerated constant to this enum **DDS_TypeCode** (p. 1428).*

- **DDS_UnsignedLong add_member_to_union** (const char * **name**, **DDS_Long** id, const **DDS_LongSeq** &labels, const **DDS_TypeCode** *tc, **DDS_Boolean** is_pointer, **DDS_ExceptionCode_t** &ex)

  *Add a new member to a union **DDS_TypeCode** (p. 1428).*

- **DDS_UnsignedLong add_member** (const char * **name**, **DDS_Long** id, const **DDS_TypeCode** *tc, **DDS_Octet** member_flags, **DDS_ExceptionCode_t** &ex)

  *Add a new member to this **DDS_TypeCode** (p. 1428).*

- **DDS_UnsignedLong add_member_ex** (const char * **name**, **DDS_Long** id, const **DDS_TypeCode** *tc, **DDS_Octet** member_flags, **DDS_Visibility** visibility, **DDS_Boolean** is_pointer, **DDS_Short** bits, **DDS_Boolean** is_must_understand, **DDS_ExceptionCode_t** &ex)

  *Add a new member to this **DDS_TypeCode** (p. 1428).*
- const **DDS_AnnotationParameterValue** * **default_value** (**DDS_ExceptionCode_t** &ex)

  *Returns the default annotation for this **DDS_TypeCode** (p. 1428).*
- const **DDS_AnnotationParameterValue** * **min_value** (**DDS_ExceptionCode_t** &ex)

  *Returns the min annotation for this **DDS_TypeCode** (p. 1428).*
- const **DDS_AnnotationParameterValue** * **max_value** (**DDS_ExceptionCode_t** &ex)

  *Returns the max annotation for this **DDS_TypeCode** (p. 1428).*
- const char * **unit** (**DDS_ExceptionCode_t** &ex)

  *Returns the unit annotation for this **DDS_TypeCode** (p. 1428).*
- const **DDS_AnnotationParameterValue** * **member_default_value** (**DDS_UnsignedLong** index, **DDS_ExceptionCode_t** &ex)

  *Returns the member default annotation for this **DDS_TypeCode** (p. 1428).*
- const **DDS_AnnotationParameterValue** * **member_min_value** (**DDS_UnsignedLong** index, **DDS_ExceptionCode_t** &ex)

  *Returns the member min annotation for this **DDS_TypeCode** (p. 1428).*
- const **DDS_AnnotationParameterValue** * **member_max_value** (**DDS_UnsignedLong** index, **DDS_ExceptionCode_t** &ex)

  *Returns the member max annotation for this **DDS_TypeCode** (p. 1428).*
- const char * **member_unit** (**DDS_UnsignedLong** index, **DDS_ExceptionCode_t** &ex)

  *Returns the unit annotation for a member of this **DDS_TypeCode** (p. 1428).*
- **DDS_Boolean is_member_must_understand** (**DDS_UnsignedLong** index, **DDS_ExceptionCode_t** &ex)

  *Returns the "must understand" property for a member of this **DDS_TypeCode** (p. 1428).*
- void **print_IDL** (**DDS_UnsignedLong** indent, **DDS_ExceptionCode_t** &ex) const

  *Prints a **DDS_TypeCode** (p. 1428) in IDL notation.*
- void **print** (const **DDS_TypeCodePrintFormatProperty** &format, **DDS_ExceptionCode_t** &ex) const

  *Prints a **DDS_TypeCode** (p. 1428).*
- void **to_string** (char *str, **DDS_UnsignedLong** &str_size, **DDS_ExceptionCode_t** &ex) const

  *Get a string representation of this **DDS_TypeCode** (p. 1428) object using the default values for **DDS_TypeCodePrintFormatProperty** (p. 1490).*
- void **to_string** (char *str, **DDS_UnsignedLong** &str_size, const **DDS_TypeCodePrintFormatProperty** &format, **DDS_ExceptionCode_t** &ex) const

  *Get a string representation of this **DDS_TypeCode** (p. 1428) object using the format described by **DDS_TypeCodePrintFormatProperty** (p. 1490).*

9.214.1 Detailed Description

The definition of a particular data type, which you can use to inspect the name, members, and other properties of types generated with rtiddsgen (see the `Code Generator User's Manual`) or to modify types you define yourself at runtime.

You create **DDS_TypeCode** (p. 1428) objects using the **DDS_TypeCodeFactory** (p. 1476) singleton. Then you can use the methods on *this* class to inspect and modify the data type definition.

This class is based on a similar class from CORBA.

MT Safety:

SAFE for read-only access, UNSAFE for modification. Modifying a single **DDS_TypeCode** (p. 1428) object concurrently from multiple threads is *unsafe*. Modifying a **DDS_TypeCode** (p. 1428) from a single thread while concurrently reading the state of that **DDS_TypeCode** (p. 1428) from another thread is also *unsafe*. However, reading the state of a **DDS_TypeCode** (p. 1428) concurrently from multiple threads, without any modification, is *safe*.

Examples

HelloWorld.cxx, and **HelloWorldPlugin.cxx**.

9.214.2 Member Function Documentation

9.214.2.1 kind()

```
DDS_TCKind DDS_TypeCode::kind (
            DDS_ExceptionCode_t & ex ) const
```

Gets the **DDS_TCKind** (p. 89) value of a type code.

Parameters

ex	<<***out***>> *(p. 433)* Parameter for error indications. The values that it can take are: • **DDS_NO_EXCEPTION_CODE** (p. 530) • **DDS_BAD_TYPECODE_SYSTEM_EXCEPTION_CODE** (p. 530)

Retrieves the kind of this **DDS_TypeCode** (p. 1428) object. The kind of a type code determines which **DDS_TypeCode** (p. 1428) methods may legally be invoked on it.

MT Safety:

SAFE.

Returns

The type code kind.

9.214.2.2 extensibility_kind()

```
DDS_ExtensibilityKind DDS_TypeCode::extensibility_kind (
            DDS_ExceptionCode_t & ex ) const
```

Gets the **DDS_ExtensibilityKind** (p. 89) value of a type code.

Parameters

ex	<<***out***>> *(p. 433)* Parameter for error indications. The values that it can take are: • **DDS_NO_EXCEPTION_CODE** (p. 530) • **DDS_BAD_TYPECODE_SYSTEM_EXCEPTION_CODE** (p. 530)

Retrieves the extensibility kind of this **DDS_TypeCode** (p. 1428) object.

In some cases, it is desirable for types to evolve without breaking interoperability with deployed components already using those types. For example:

- A new set of applications to be integrated into an existing system may want to introduce additional fields into a structure. These new fields can be safely ignored by already deployed applications, but applications that do understand the new fields can benefit from their presence.
- A new set of applications to be integrated into an existing system may want to increase the maximum size of some sequence or string in a Type. Existing applications can receive data samples from these new applications as long as the actual number of elements (or length of the strings) in the received data sample does not exceed what the receiving applications expects. If a received data sample exceeds the limits expected by the receiving application, then the sample can be safely ignored (filtered out) by the receiver.

In order to support use cases such as these, the type system introduces the concept of appendable and mutable types.

- A type may be final, indicating that the range of its possible data values is strictly defined. In particular, it is not possible to add elements to members of collection or aggregated types while maintaining type assignability.
- A type may be appendable, indicating that two types, where one contains all of the elements/members of the other plus additional elements/members appended to the end, may remain assignable.
- A type may be mutable, indicating that two types may differ from one another in the additional, removal, and/or transposition of elements/members while remaining assignable.

The extensibility of **DDS_TK_STRUCT** (p. 89), **DDS_TK_UNION** (p. 89), **DDS_TK_VALUE** (p. 89), and **DDS_TK_ENUM** (p. 89) can be change using the built-in "Extensibility" annotation when the type is declared.

IDL examples:

```
@final

struct MyType {

    long member_1;

}
@appendable

struct MyType {

    long member_1;

}
@mutable

struct MyType {
```

```
    long member_1;

}
```

XML example:

```
<struct name="MyType" extensibility="final">

    <member name="member_1" type="long"/>

</struct>
<struct name="MyType" extensibility="appendable">

    <member name="member_1" type="long"/>

</struct>
<struct name="MyType" extensibility="mutable">

    <member name="member_1" type="long"/>

</struct>
```

XSD example:

```
<xsd:complexType name="MyType">

    <xsd:sequence>

        <xsd:element name="member_1" minOccurs="1" maxOccurs="1" type="xsd:int"/>

    </xsd:sequence>

</xsd:complexType>

<!-- @struct true -->

<!-- @extensibility FINAL_EXTENSIBILITY -->
<xsd:complexType name="MyType">

    <xsd:sequence>

        <xsd:element name="member_1" minOccurs="1" maxOccurs="1" type="xsd:int"/>

    </xsd:sequence>

</xsd:complexType>

<!-- @struct true -->

<!-- @extensibility EXTENSIBLE_EXTENSIBILITY -->
<xsd:complexType name="MyType">

    <xsd:sequence>

        <xsd:element name="member_1" minOccurs="1" maxOccurs="1" type="xsd:int"/>

    </xsd:sequence>

</xsd:complexType>

<!-- @struct true -->

<!-- @extensibility MUTABLE_EXTENSIBILITY -->
```

For TypeCodes built at run-time using the **DDS_TypeCodeFactory** (p. 1476) API, the extensibility can be provided as a parameter of the following APIs:

- **DDS_TypeCodeFactory::create_struct_tc** (p. 1480)
- **DDS_TypeCodeFactory::create_value_tc** (p. 1482)
- **DDS_TypeCodeFactory::create_union_tc** (p. 1483)
- **DDS_TypeCodeFactory::create_enum_tc** (p. 1485)

See also

DDS_ExtensibilityKind (p. 89)

MT Safety:

SAFE.

Returns

The type code extensibility kind.

9.214.2.3 equal()

```
DDS_Boolean DDS_TypeCode::equal (
            const  DDS_TypeCode * tc,
             DDS_ExceptionCode_t & ex ) const
```

Compares two **DDS_TypeCode** (p. 1428) objects for equality.

MT Safety:

SAFE.

For equality and assignability purposes, **DDS_TK_STRUCT** (p. 89) and **DDS_TK_VALUE** (p. 89) are considered equivalent.

The **DDS_TypeCode** (p. 1428) of structs inheriting from other structs has a **DDS_TK_VALUE** (p. 89) kind.

For example:

```
struct MyStruct: MyBaseStruct {

    long member_1;

};
```

The code generation for the previous type will generate a **DDS_TypeCode** (p. 1428) with **DDS_TK_VALUE** (p. 89) kind.

Additionally, for equality and assignability purposes, **DDS_TK_OCTET** (p. 89), **DDS_TK_INT8** (p. 89), and **DDS_TK_↩ UINT8** (p. 89) are considered equivalent.

Parameters

tc	<<***in***>> *(p. 433)* Type code that will be compared with this **DDS_TypeCode** (p. 1428).
ex	<<***out***>> *(p. 433)* Parameter for error indications. The values that can take are: • **DDS_NO_EXCEPTION_CODE** (p. 530) • **DDS_BAD_PARAM_SYSTEM_EXCEPTION_CODE** (p. 530) • **DDS_BAD_TYPECODE_SYSTEM_EXCEPTION_CODE** (p. 530)

Returns

DDS_BOOLEAN_TRUE (p. 513) if the type codes are equal. Otherwise, **DDS_BOOLEAN_FALSE** (p. 513).

9.214.2.4 is_assignable_from()

```
DDS_Boolean DDS_TypeCode::is_assignable_from (
            const  DDS_TypeCode * from,
             DDS_ExceptionCode_t & ex ) const
```

Check if this type is assignable from another type.

If this type is assignable from the `from` type, it means that it is possible, in a structural way, to set the contents of an object of this type to the contents of an object of type `from` (or a subset of those contents) without leading to incorrect interpretations of that information.

For additional information on type assignability, refer to the `OMG Extensible and Dynamic Topic Types for DDS Specification`.

MT Safety:

SAFE.

Parameters

from	<<***in***>> *(*p. *433)* The type to check assignability from.
ex	<<***out***>> *(*p. *433)* Parameter for error indications.

Returns

DDS_BOOLEAN_TRUE (p. 513) if this type is assignable from `from`. Otherwise, **DDS_BOOLEAN_FALSE** (p. 513).

9.214.2.5 name()

```
const char * DDS_TypeCode::name (
             DDS_ExceptionCode_t & ex ) const
```

Retrieves the simple name identifying this **DDS_TypeCode** (p. 1428) object within its enclosing scope.

Precondition

self kind is **DDS_TK_STRUCT** (p. 89), **DDS_TK_UNION** (p. 89), **DDS_TK_ENUM** (p. 89), **DDS_TK_VALUE** (p. 89), or **DDS_TK_ALIAS** (p. 89).

MT Safety:

SAFE.

Parameters

ex	<<***out***>> *(p. 433)* Parameter for error indications. The values that can take are: • **DDS_NO_EXCEPTION_CODE** (p. 530) • **DDS_BAD_TYPECODE_SYSTEM_EXCEPTION_CODE** (p. 530) • **DDS_BADKIND_USER_EXCEPTION_CODE** (p. 530)

Returns

Name of the type code if no errors.

9.214.2.6 member_count()

```
DDS_UnsignedLong DDS_TypeCode::member_count (
            DDS_ExceptionCode_t & ex ) const
```

Returns the number of members of the type code.

The method member_count can be invoked on structure, union, and enumeration **DDS_TypeCode** (p. 1428) objects.

Precondition

self kind is **DDS_TK_STRUCT** (p. 89), **DDS_TK_UNION** (p. 89), **DDS_TK_ENUM** (p. 89) or **DDS_TK_VALUE** (p. 89).

MT Safety:

SAFE.

Parameters

ex	<<***out***>> *(p. 433)* Parameter for error indications. The values that can take are: • **DDS_NO_EXCEPTION_CODE** (p. 530) • **DDS_BAD_TYPECODE_SYSTEM_EXCEPTION_CODE** (p. 530) • **DDS_BADKIND_USER_EXCEPTION_CODE** (p. 530)

Returns

The number of members constituting the type described by this **DDS_TypeCode** (p. 1428) object if no errors.

9.214.2.7 member_name()

```
const char * DDS_TypeCode::member_name (
              DDS_UnsignedLong index,
              DDS_ExceptionCode_t & ex ) const
```

Returns the name of a type code member identified by the given index.

The method member_name can be invoked on structure, union, and enumeration **DDS_TypeCode** (p. 1428) objects.

Precondition

self kind is **DDS_TK_STRUCT** (p. 89), **DDS_TK_UNION** (p. 89), **DDS_TK_ENUM** (p. 89) or **DDS_TK_VALUE** (p. 89).
The index param must be in the interval [0,(member count-1)].

MT Safety:

SAFE.

Parameters

index	<<***in***>> *(p. 433)* Member index in the interval [0,(member count-1)].
ex	<<***out***>> *(p. 433)* Parameter for error indications. The values that can take are: • **DDS_NO_EXCEPTION_CODE** (p. 530) • **DDS_BAD_TYPECODE_SYSTEM_EXCEPTION_CODE** (p. 530) • **DDS_BADKIND_USER_EXCEPTION_CODE** (p. 530)
	• **DDS_BOUNDS_USER_EXCEPTION_CODE** (p. 530)

Returns

Name of the member if no errors.

9.214.2.8 find_member_by_name()

```
DDS_UnsignedLong DDS_TypeCode::find_member_by_name (
            const char * name,
             DDS_ExceptionCode_t & ex ) const
```

Get the index of the member of the given name.

This method is applicable to **DDS_TypeCode** (p. 1428) objects representing structs (**DDS_TK_STRUCT** (p. 89)) and union (**DDS_TK_UNION** (p. 89)) types.

Parameters

name	*<<**in**>> (p. 433)* The member name.
ex	*<<**out**>> (p. 433)* Parameter for error indications. The values that can take are: • **DDS_NO_EXCEPTION_CODE** (p. 530) • **DDS_BAD_TYPECODE_SYSTEM_EXCEPTION_CODE** (p. 530) • **DDS_BADKIND_USER_EXCEPTION_CODE** (p. 530)

Returns

The index of the member of the given name or **DDS_TYPECODE_INDEX_INVALID** (p. 83) if the member is not found.

MT Safety:

SAFE.

9.214.2.9 member_type()

```
DDS_TypeCode * DDS_TypeCode::member_type (
            DDS_UnsignedLong index,
            DDS_ExceptionCode_t & ex ) const
```

Retrieves the **DDS_TypeCode** (p. 1428) object describing the type of the member identified by the given index.

The method member_type can be invoked on structure and union type codes.

Precondition

self kind is **DDS_TK_STRUCT** (p. 89), **DDS_TK_UNION** (p. 89) or **DDS_TK_VALUE** (p. 89).
The index param must be in the interval [0,(member count-1)].

MT Safety:

SAFE.

Parameters

index	<<***in***>> *(p. 433)* Member index in the interval [0,(member count-1)].
ex	<<***out***>> *(p. 433)* Parameter for error indications. The values that can take are: • **DDS_NO_EXCEPTION_CODE** (p. 530) • **DDS_BAD_TYPECODE_SYSTEM_EXCEPTION_CODE** (p. 530) • **DDS_BADKIND_USER_EXCEPTION_CODE** (p. 530) • **DDS_BOUNDS_USER_EXCEPTION_CODE** (p. 530)

Returns

The **DDS_TypeCode** (p. 1428) object describing the member at the given index if no errors.

9.214.2.10 member_label_count()

```
DDS_UnsignedLong DDS_TypeCode::member_label_count (
            DDS_UnsignedLong index,
            DDS_ExceptionCode_t & ex ) const
```

Returns the number of labels associated to the index-th union member.

The method can be invoked on union **DDS_TypeCode** (p. 1428) objects.

This function is an RTI Connext extension to the CORBA Type Code Specification.

Precondition

self kind is **DDS_TK_UNION** (p. 89).
The index param must be in the interval [0,(member count-1)].

MT Safety:

SAFE.

Parameters

index	<<***in***>> *(*p. *433)* Member index in the interval [0,(member count-1)].
ex	<<***out***>> *(*p. *433)* Parameter for error indications. The values that can take are: • **DDS_NO_EXCEPTION_CODE** (p. 530) • **DDS_BAD_TYPECODE_SYSTEM_EXCEPTION_CODE** (p. 530) • **DDS_BADKIND_USER_EXCEPTION_CODE** (p. 530) • **DDS_BOUNDS_USER_EXCEPTION_CODE** (p. 530)

Returns

Number of labels if no errors.

9.214.2.11 member_label()

```
DDS_Long DDS_TypeCode::member_label (
            DDS_UnsignedLong member_index,
            DDS_UnsignedLong label_index,
            DDS_ExceptionCode_t & ex ) const
```

Return the label_index-th label associated to the member_index-th member.

This method has been modified for RTI Connext from the CORBA Type code Specification.

Example:
case 1: Label index 0
case 2: Label index 1
short short_member;

The method can be invoked on union **DDS_TypeCode** (p. 1428) objects.

Precondition

self kind is **DDS_TK_UNION** (p. 89).
The member_index param must be in the interval [0,(member count-1)].
The label_index param must be in the interval [0,(member labels count-1)].

MT Safety:

SAFE.

Parameters

member_index	<<***in***>> *(*p. *433)* Member index.
label_index	<<***in***>> *(*p. *433)* Label index.
ex	<<***out***>> *(*p. *433)* Parameter for error indications. The values that can take are: • **DDS_NO_EXCEPTION_CODE** (p. 530) • **DDS_BAD_TYPECODE_SYSTEM_EXCEPTION_CODE** (p. 530) • **DDS_BADKIND_USER_EXCEPTION_CODE** (p. 530) • **DDS_BOUNDS_USER_EXCEPTION_CODE** (p. 530)

Returns

The evaluated value of the label if no errors.

9.214.2.12 member_ordinal()

```
DDS_Long DDS_TypeCode::member_ordinal (
            DDS_UnsignedLong index,
            DDS_ExceptionCode_t & ex ) const
```

Returns the ordinal that corresponds to the index-th enum value.

The method can be invoked on enum **DDS_TypeCode** (p. 1428) objects.

This function is an RTI Connext extension to the CORBA Type Code Specification.

Precondition

self kind is **DDS_TK_ENUM** (p. 89).
Member index in the interval [0,(member count-1)].

MT Safety:

SAFE.

Parameters

index	<<***in***>> *(*p. *433)* Member index in the interval [0,(member count-1)].
ex	<<***out***>> *(*p. *433)* Parameter for error indications. The values that can take are: • **DDS_NO_EXCEPTION_CODE** (p. 530) • **DDS_BAD_TYPECODE_SYSTEM_EXCEPTION_CODE** (p. 530) • **DDS_BADKIND_USER_EXCEPTION_CODE** (p. 530)

Returns

Ordinal that corresponds to the index-th enumerator if no errors.

9.214.2.13 is_member_key()

```
DDS_Boolean DDS_TypeCode::is_member_key (
            DDS_UnsignedLong index,
            DDS_ExceptionCode_t & ex ) const
```

Function that tells if a member is a key or not.

This function is an RTI Connext extension to the CORBA Type Code Specification.

Precondition

self kind is **DDS_TK_STRUCT** (p. 89) or **DDS_TK_VALUE** (p. 89).
The index param must be in the interval [0,(member count-1)].

MT Safety:

SAFE.

Parameters

index	<<***in***>> *(p. 433)* Member index in the interval [0,(member count-1)].
ex	<<***out***>> *(p. 433)* Parameter for error indications. The values that can take are: • **DDS_NO_EXCEPTION_CODE** (p. 530) • **DDS_BAD_TYPECODE_SYSTEM_EXCEPTION_CODE** (p. 530) • **DDS_BADKIND_USER_EXCEPTION_CODE** (p. 530) • **DDS_BOUNDS_USER_EXCEPTION_CODE** (p. 530)

Returns

DDS_BOOLEAN_TRUE (p. 513) if the member is a key. Otherwise, **DDS_BOOLEAN_FALSE** (p. 513).

9.214.2.14 is_member_required()

```
DDS_Boolean DDS_TypeCode::is_member_required (
            DDS_UnsignedLong index,
            DDS_ExceptionCode_t & ex ) const
```

Indicates whether a given member of a type is required to be present in every sample of that type.

A non-key member is required if it has not been marked as optional. All key members are required.

See also

DDS_TYPECODE_NONKEY_MEMBER (p. 85)
DDS_TYPECODE_NONKEY_REQUIRED_MEMBER (p. 86)

MT Safety:

SAFE.

9.214.2.15 is_member_pointer()

```
DDS_Boolean DDS_TypeCode::is_member_pointer (
            DDS_UnsignedLong index,
            DDS_ExceptionCode_t & ex ) const
```

Function that tells if a member is a pointer or not.

The method is_member_pointer can be invoked on union and structs type objects

This function is an RTI Connext extension to the CORBA Type Code Specification.

Precondition

self kind is **DDS_TK_STRUCT** (p. 89), **DDS_TK_UNION** (p. 89) or **DDS_TK_VALUE** (p. 89).
The index param must be in the interval [0,(member count-1)].

MT Safety:

SAFE.

Parameters

index	<<***in***>> *(p. 433)* Index of the member for which type information is begin requested.
ex	<<***out***>> *(p. 433)* Parameter for error indications. The values that can take are: • **DDS_NO_EXCEPTION_CODE** (p. 530) • **DDS_BAD_TYPECODE_SYSTEM_EXCEPTION_CODE** (p. 530) • **DDS_BADKIND_USER_EXCEPTION_CODE** (p. 530)

Returns

DDS_BOOLEAN_TRUE (p. 513) if the member is a pointer. Otherwise, **DDS_BOOLEAN_FALSE** (p. 513).

9.214.2.16 is_member_bitfield()

```
DDS_Boolean DDS_TypeCode::is_member_bitfield (
            DDS_UnsignedLong index,
            DDS_ExceptionCode_t & ex ) const
```

Function that tells if a member is a bitfield or not.

The method can be invoked on struct type objects.

This function is an RTI Connext extension to the CORBA Type Code Specification.

Precondition

self kind is **DDS_TK_STRUCT** (p. 89) or **DDS_TK_VALUE** (p. 89).
The index param must be in the interval [0,(member count-1)].

MT Safety:

SAFE.

Parameters

index	<<***in***>> *(p. 433)* Member index in the interval [0,(member count-1)].
ex	<<***out***>> *(p. 433)* Parameter for error indications. The values that can take are: • **DDS_NO_EXCEPTION_CODE** (p. 530) • **DDS_BAD_TYPECODE_SYSTEM_EXCEPTION_CODE** (p. 530) • **DDS_BADKIND_USER_EXCEPTION_CODE** (p. 530) • **DDS_BOUNDS_USER_EXCEPTION_CODE** (p. 530)

Returns

DDS_BOOLEAN_TRUE (p. 513) if the member is a bitfield. Otherwise, **DDS_BOOLEAN_FALSE** (p. 513).

9.214.2.17 member_bitfield_bits()

```
DDS_Short DDS_TypeCode::member_bitfield_bits (
            DDS_UnsignedLong index,
            DDS_ExceptionCode_t & ex ) const
```

Returns the number of bits of a bitfield member.

The method can be invoked on struct type objects.

This function is an RTI Connext extension to the CORBA Type Code Specification.

Precondition

self kind is **DDS_TK_STRUCT** (p. 89) or **DDS_TK_VALUE** (p. 89).
The index param must be in the interval [0,(member count-1)].

MT Safety:

SAFE.

Parameters

index	<<***in***>> *(p. 433)* Member index in the interval [0,(member count-1)].
ex	<<***out***>> *(p. 433)* Parameter for error indications. The values that can take are: • **DDS_NO_EXCEPTION_CODE** (p. 530) • **DDS_BAD_TYPECODE_SYSTEM_EXCEPTION_CODE** (p. 530) • **DDS_BADKIND_USER_EXCEPTION_CODE** (p. 530) • **DDS_BOUNDS_USER_EXCEPTION_CODE** (p. 530)

Returns

The number of bits of the bitfield or **DDS_TYPECODE_NOT_BITFIELD** (p. 83) if the member is not a bitfield.

9.214.2.18 member_visibility()

```
DDS_Visibility DDS_TypeCode::member_visibility (
            DDS_UnsignedLong index,
            DDS_ExceptionCode_t & ex ) const
```

Returns the constant that indicates the visibility of the index-th member.

Precondition

self kind is **DDS_TK_VALUE** (p. 89) or **DDS_TK_STRUCT** (p. 89). The index param must be in the interval [0,(member count-1)].

MT Safety:

SAFE.

For **DDS_TK_STRUCT** (p. 89), this method always returns **DDS_PUBLIC_MEMBER** (p. 85).

Parameters

index	<<***in***>> *(p. 433)* Member index in the interval [0,(member count-1)].
ex	<<***out***>> *(p. 433)* Parameter for error indications. The values that can take are: • **DDS_NO_EXCEPTION_CODE** (p. 530) • **DDS_BAD_TYPECODE_SYSTEM_EXCEPTION_CODE** (p. 530) • **DDS_BADKIND_USER_EXCEPTION_CODE** (p. 530) • **DDS_BOUNDS_USER_EXCEPTION_CODE** (p. 530)

Returns

One of the following constants: **DDS_PRIVATE_MEMBER** (p. 85) or **DDS_PUBLIC_MEMBER** (p. 85).

9.214.2.19 discriminator_type()

```
DDS_TypeCode * DDS_TypeCode::discriminator_type (
            DDS_ExceptionCode_t & ex ) const
```

Returns the discriminator type code.

The method discriminator_type can be invoked only on union **DDS_TypeCode** (p. 1428) objects.

Precondition

self kind is **DDS_TK_UNION** (p. 89).

MT Safety:

SAFE.

Parameters

ex	<<***out***>> *(p. 433)* Parameter for error indications. The values that can take are: • **DDS_NO_EXCEPTION_CODE** (p. 530) • **DDS_BAD_TYPECODE_SYSTEM_EXCEPTION_CODE** (p. 530) • **DDS_BADKIND_USER_EXCEPTION_CODE** (p. 530)

Returns

DDS_TypeCode (p. 1428) object describing the discriminator of the union type if no errors.

9.214.2.20 length()

```
DDS_UnsignedLong DDS_TypeCode::length (
            DDS_ExceptionCode_t & ex ) const
```

Returns the number of elements in the type described by this type code.

Length is:

- The maximum length of the string for string type codes.
- The maximum length of the sequence for sequence type codes.
- The first dimension of the array for array type codes.

Precondition

self kind is **DDS_TK_ARRAY** (p. 89), **DDS_TK_SEQUENCE** (p. 89), **DDS_TK_STRING** (p. 89) or **DDS_TK_↩ WSTRING** (p. 89).

MT Safety:

SAFE.

Parameters

ex	<<***out***>> *(p. 433)* Parameter for error indications. The values that can take are: • **DDS_NO_EXCEPTION_CODE** (p. 530) • **DDS_BAD_TYPECODE_SYSTEM_EXCEPTION_CODE** (p. 530) • **DDS_BADKIND_USER_EXCEPTION_CODE** (p. 530)

Returns

The bound for strings and sequences, or the number of elements for arrays if no errors.

9.214.2.21 array_dimension_count()

```
DDS_UnsignedLong DDS_TypeCode::array_dimension_count (
            DDS_ExceptionCode_t & ex ) const
```

This function returns the number of dimensions of an array type code.

This function is an RTI Connext extension to the CORBA Type Code Specification.

Precondition

self kind is **DDS_TK_ARRAY** (p. 89).

MT Safety:

SAFE.

Parameters

ex	<<***out***>> *(p. 433)* Parameter for error indications. The values that can take are: • **DDS_NO_EXCEPTION_CODE** (p. 530) • **DDS_BAD_TYPECODE_SYSTEM_EXCEPTION_CODE** (p. 530) • **DDS_BADKIND_USER_EXCEPTION_CODE** (p. 530)

Returns

Number of dimensions if no errors.

9.214.2.22 array_dimension()

```
DDS_UnsignedLong DDS_TypeCode::array_dimension (
            DDS_UnsignedLong index,
            DDS_ExceptionCode_t & ex ) const
```

This function returns the index-th dimension of an array type code.

This function is an RTI Connext extension to the CORBA Type Code Specification.

Precondition

self kind is **DDS_TK_ARRAY** (p. 89).
Dimension index in the interval [0,(dimensions count-1)].

MT Safety:

SAFE.

Parameters

index	<<***in***>> *(p. 433)* Dimension index in the interval [0,(dimensions count-1)].
ex	<<***out***>> *(p. 433)* Parameter for error indications. The values that can take are: • **DDS_NO_EXCEPTION_CODE** (p. 530) • **DDS_BAD_TYPECODE_SYSTEM_EXCEPTION_CODE** (p. 530) • **DDS_BADKIND_USER_EXCEPTION_CODE** (p. 530) • **DDS_BOUNDS_USER_EXCEPTION_CODE** (p. 530)

Returns

Requested dimension if no errors.

9.214.2.23 element_count()

```
DDS_UnsignedLong DDS_TypeCode::element_count (
            DDS_ExceptionCode_t & ex ) const
```

The number of elements in an array.

This operation isn't relevant for other kinds of types.

MT Safety:

SAFE.

9.214.2.24 content_type()

```
DDS_TypeCode * DDS_TypeCode::content_type (
            DDS_ExceptionCode_t & ex ) const
```

Returns the **DDS_TypeCode** (p. 1428) object representing the type for the members of the object described by this **DDS_TypeCode** (p. 1428) object.

For sequences and arrays, it returns the element type. For aliases, it returns the original type.

Precondition

self kind is **DDS_TK_ARRAY** (p. 89), **DDS_TK_SEQUENCE** (p. 89) or **DDS_TK_ALIAS** (p. 89).

MT Safety:

SAFE.

Parameters

ex	<<***out***>> *(p. 433)* Parameter for error indications. The values that can take are: • **DDS_NO_EXCEPTION_CODE** (p. 530) • **DDS_BAD_TYPECODE_SYSTEM_EXCEPTION_CODE** (p. 530) • **DDS_BADKIND_USER_EXCEPTION_CODE** (p. 530)

Returns

A **DDS_TypeCode** (p. 1428) object representing the element type for sequences and arrays, and the original type for aliases.

9.214.2.25 is_alias_pointer()

```
DDS_Boolean DDS_TypeCode::is_alias_pointer (
            DDS_ExceptionCode_t & ex ) const
```

Function that tells if an alias is a pointer or not.

This function is an RTI Connext extension to the CORBA Type Code Specification.

Precondition

self kind is **DDS_TK_ALIAS** (p. 89).

MT Safety:

SAFE.

Parameters

ex	<<***out***>> *(p. 433)* Parameter for error indications. The values that can take are: • **DDS_NO_EXCEPTION_CODE** (p. 530) • **DDS_BAD_TYPECODE_SYSTEM_EXCEPTION_CODE** (p. 530) • **DDS_BADKIND_USER_EXCEPTION_CODE** (p. 530)

Returns

DDS_BOOLEAN_TRUE (p. 513) if an alias is a pointer to the aliased type. Otherwise, **DDS_BOOLEAN_FALSE** (p. 513).

9.214.2.26 default_index()

```
DDS_Long DDS_TypeCode::default_index (
            DDS_ExceptionCode_t & ex ) const
```

Returns the index of the default member, or -1 if there is no default member.

The method default_index can be invoked only on union **DDS_TypeCode** (p. 1428) objects.

Precondition

self kind is **DDS_TK_UNION** (p. 89)

MT Safety:

SAFE.

Parameters

ex	<<***out***>> *(p. 433)* Parameter for error indications. The values that can take are: • **DDS_NO_EXCEPTION_CODE** (p. 530) • **DDS_BAD_TYPECODE_SYSTEM_EXCEPTION_CODE** (p. 530) • **DDS_BADKIND_USER_EXCEPTION_CODE** (p. 530)

Returns

The index of the default member, or -1 if there is no default member.

9.214.2.27 concrete_base_type()

```
DDS_TypeCode * DDS_TypeCode::concrete_base_type (
            DDS_ExceptionCode_t & ex ) const
```

Returns the **DDS_TypeCode** (p. 1428) that describes the concrete base type of the value type that this **DDS_TypeCode** (p. 1428) object describes.

Precondition

self kind is **DDS_TK_VALUE** (p. 89) or **DDS_TK_STRUCT** (p. 89).

MT Safety:

SAFE.

For **DDS_TK_STRUCT** (p. 89), this method always returns **DDS_TK_NULL** (p. 89).

The **DDS_TypeCode** (p. 1428) of structs inheriting from other structs has a **DDS_TK_VALUE** (p. 89) kind.

For example:

```
struct MyStruct: MyBaseStruct {

    long member_1;

};
```

The code generation for the previous type will generate a **DDS_TypeCode** (p. 1428) with **DDS_TK_VALUE** (p. 89) kind.

Parameters

ex	<<***out***>> *(p. 433)* Parameter for error indications. The values that can take are: • **DDS_NO_EXCEPTION_CODE** (p. 530) • **DDS_BAD_TYPECODE_SYSTEM_EXCEPTION_CODE** (p. 530) • **DDS_BADKIND_USER_EXCEPTION_CODE** (p. 530)

Returns

DDS_TypeCode (p. 1428) that describes the concrete base type or NULL if there is no a concrete base type.

9.214.2.28 type_modifier()

```
DDS_ValueModifier DDS_TypeCode::type_modifier (
            DDS_ExceptionCode_t & ex ) const
```

Returns a constant indicating the modifier of the value type that this **DDS_TypeCode** (p. 1428) object describes.

Precondition

self kind is **DDS_TK_VALUE** (p. 89) or **DDS_TK_STRUCT** (p. 89).

For **DDS_TK_STRUCT** (p. 89), this method always returns **DDS_VM_NONE** (p. 84).

MT Safety:

SAFE.

Parameters

ex	<<***out***>> *(p. 433)* Parameter for error indications. The values that can take are: • **DDS_NO_EXCEPTION_CODE** (p. 530) • **DDS_BAD_TYPECODE_SYSTEM_EXCEPTION_CODE** (p. 530) • **DDS_BADKIND_USER_EXCEPTION_CODE** (p. 530)

Returns

One of the following type modifiers: **DDS_VM_NONE** (p. 84), **DDS_VM_ABSTRACT** (p. 84), **DDS_VM_CUSTOM** (p. 84) or **DDS_VM_TRUNCATABLE** (p. 84).

9.214.2.29 member_id()

```
DDS_Long DDS_TypeCode::member_id (
            DDS_UnsignedLong index,
            DDS_ExceptionCode_t & ex ) const
```

Returns the ID of the TypeCode member identified by the given index.

This function is an RTI Connext extension to the CORBA Type Code Specification.

The method can be invoked on aggregation **DDS_TypeCode** (p. 1428) objects.

All members of aggregated types have an integral member ID that uniquely identifies them within their defining type.

In IDL, you can specify the member ID using the built-in annotation ID. For example:

```
struct MyType {

    long member_1; //@ID 200

} //@Extensibility MUTABLE_EXTENSIBILITY
```

In XML, you can specify the member ID using the attribute 'id':

```
<struct name="MyType" extensibility="mutable" id="200">

    <member name="member_1" type="long"/>

</struct>
```

In XSD, you can specify the member ID using the built-in annotation ID

```
<xsd:complexType name="MyType">

    <xsd:sequence>

        <xsd:element name="member_1" minOccurs="1" maxOccurs="1" type="xsd:int"/>

        <!-- @id 200 -->

    </xsd:sequence>

</xsd:complexType>

<!-- @extensibility MUTABLE_EXTENSIBILITY -->
```

Precondition

self kind is **DDS_TK_STRUCT** (p. 89), **DDS_TK_VALUE** (p. 89), or **DDS_TK_UNION** (p. 89)
Member index in the interval [0,(member count-1)].

MT Safety:

SAFE.

Parameters

index	<<***in***>> *(*p. *433)* Member index in the interval [0,(member count-1)].
ex	<<***out***>> *(*p. *433)* Parameter for error indications. The values that can take are: • **DDS_NO_EXCEPTION_CODE** (p. 530) • **DDS_BAD_TYPECODE_SYSTEM_EXCEPTION_CODE** (p. 530) • **DDS_BADKIND_USER_EXCEPTION_CODE** (p. 530) • **DDS_BOUNDS_USER_EXCEPTION_CODE** (p. 530)

Returns

ID of the member if no errors.

9.214.2.30 find_member_by_id()

```
DDS_UnsignedLong DDS_TypeCode::find_member_by_id (
            DDS_Long id,
            DDS_ExceptionCode_t & ex ) const
```

Get the index of the member of the given ID.

This method is applicable to **DDS_TypeCode** (p. 1428) objects representing structs (**DDS_TK_STRUCT** (p. 89)) and union (**DDS_TK_UNION** (p. 89)) types.

Parameters

id	<<***in***>> *(*p. *433)* The member ID.
ex	<<***out***>> *(*p. *433)* Parameter for error indications. The values that can take are: • **DDS_NO_EXCEPTION_CODE** (p. 530) • **DDS_BAD_TYPECODE_SYSTEM_EXCEPTION_CODE** (p. 530) • **DDS_BADKIND_USER_EXCEPTION_CODE** (p. 530)

Returns

The index of the member of the given ID or **DDS_TYPECODE_INDEX_INVALID** (p. 83) if the member is not found.

MT Safety:

SAFE.

9.214.2.31 get_type_object_serialized_size()

```
DDS_UnsignedLong DDS_TypeCode::get_type_object_serialized_size (
            DDS_ExceptionCode_t & ex ) const
```

Gets the serialized size of the TypeObject v1 created from this **DDS_TypeCode** (p. 1428).

Parameters

ex	<<***out***>> *(p. 433)* Parameter for error indications. The values that it can take are: • **DDS_NO_EXCEPTION_CODE** (p. 530) • **DDS_SYSTEM_EXCEPTION_CODE** (p. 530)

The default buffer size used for storing a TypeObject v1 is 8192 bytes. For a large TypeObject v1, this API can be used to determine the size that needs to be set in **DDS_DomainParticipantResourceLimitsQosPolicy::type_object_max_serialized_length** (p. 994)

MT Safety:

SAFE.

Returns

The TypeObject v1 serialized size.

9.214.2.32 get_cdr_serialized_sample_max_size()

```
DDS_UnsignedLong DDS_TypeCode::get_cdr_serialized_sample_max_size (
            DDS_DataRepresentationId_t representation_id,
            DDS_ExceptionCode_t & ex ) const
```

[DEPRECATED] Gets the maximum serialized size of samples of this type

DEPRECATED: use **DDS_TypeCode::cdr_serialized_sample_max_size** (p. 1458) instead.

Obtains the maximum possible size in bytes of any serialized data sample of this type.

Precondition

The type is an aggregation type (struct, union)

Parameters

representation↩ _id	The serialized data representation for which we calculate the maximum size.
ex	<<***out***>> *(*p. *433)* Parameter for error indications.

Returns

The maximum size

9.214.2.33 cdr_serialized_sample_max_size() [1/2]

```
DDS_UnsignedLong DDS_TypeCode::cdr_serialized_sample_max_size (
            DDS_DataRepresentationId_t representation_id,
            DDS_ExceptionCode_t & ex ) const
```

Gets the maximum serialized size of samples of this type.

Obtains the maximum possible size in bytes of any serialized data sample of this type.

Precondition

The type is an aggregation type (struct, union)

Parameters

representation↩ _id	The serialized data representation for which we calculate the maximum size.
ex	<<***out***>> *(*p. *433)* Parameter for error indications.

Returns

The maximum size

9.214.2.34 cdr_serialized_sample_min_size() [1/2]

```
DDS_UnsignedLong DDS_TypeCode::cdr_serialized_sample_min_size (
            DDS_DataRepresentationId_t representation_id,
            DDS_ExceptionCode_t & ex ) const
```

Gets the minimum serialized size of samples of this type.

Obtains the minimum possible size in bytes of any serialized data sample of this type.

Precondition

The type is an aggregation type (struct, union)

Parameters

representation↩ _id	The serialized data representation for which we calculate the minimum size.
ex	<<***out***>> *(*p. *433)* Parameter for error indications.

Returns

The minimum size

9.214.2.35 cdr_serialized_sample_key_max_size() [1/2]

```
DDS_UnsignedLong DDS_TypeCode::cdr_serialized_sample_key_max_size (
            DDS_DataRepresentationId_t representation_id,
            DDS_ExceptionCode_t & ex ) const
```

Gets the maximum serialized size of sample keys of this type.

Obtains the maximum possible size in bytes of the serialized keys of any data sample of this type

Precondition

The type is an aggregation type (struct, union)

Parameters

representation↩ _id	The serialized data representation for which we calculate the maximum key size.
ex	<<***out***>> *(*p. *433)* Parameter for error indications.

Returns

The maximum key size

9.214.2.36 cdr_serialized_sample_max_size() [2/2]

```
DDS_UnsignedLong DDS_TypeCode::cdr_serialized_sample_max_size (
            DDS_ExceptionCode_t & ex ) const
```

Gets the maximum serialized size of samples of this type.

Obtains the maximum possible size in bytes of any serialized data sample of this type.

Assumes the **DDS_DataRepresentationId_t** (p. 568) to be DDS_AUTO_DATA_REPRESENTATION

Precondition

The type is an aggregation type (struct, union)

Parameters

ex	<<***out***>> *(p. 433)* Parameter for error indications.

Returns

The maximum size

9.214.2.37 cdr_serialized_sample_min_size() [2/2]

```
DDS_UnsignedLong DDS_TypeCode::cdr_serialized_sample_min_size (
            DDS_ExceptionCode_t & ex ) const
```

Gets the minimum serialized size of samples of this type.

Obtains the minimum possible size in bytes of any serialized data sample of this type.

Assumes the **DDS_DataRepresentationId_t** (p. 568) to be DDS_AUTO_DATA_REPRESENTATION

Precondition

The type is an aggregation type (struct, union)

Parameters

ex	<<***out***>> *(p. 433)* Parameter for error indications.

Returns

The minimum size

9.214.2.38 cdr_serialized_sample_key_max_size() [2/2]

```
DDS_UnsignedLong DDS_TypeCode::cdr_serialized_sample_key_max_size (
            DDS_ExceptionCode_t & ex ) const
```

Gets the maximum serialized size of sample keys of this type.

Obtains the maximum possible size in bytes of the serialized keys of any data sample of this type

Assumes the **DDS_DataRepresentationId_t** (p. 568) to be DDS_AUTO_DATA_REPRESENTATION

Precondition

The type is an aggregation type (struct, union)

Parameters

ex	<<***out***>> *(p. 433)* Parameter for error indications.

Returns

The maximum key size

9.214.2.39 add_member_to_enum()

```
DDS_UnsignedLong DDS_TypeCode::add_member_to_enum (
            const char * name,
             DDS_Long ordinal,
             DDS_ExceptionCode_t & ex )
```

Add a new enumerated constant to this enum **DDS_TypeCode** (p. 1428).

This method is applicable to **DDS_TypeCode** (p. 1428) objects representing enumerations (**DDS_TK_ENUM** (p. 89)). To add a field to a structured type, see **DDS_TypeCode::add_member_to_enum** (p. 1461).

Modifying a **DDS_TypeCode** (p. 1428) – such as by adding a member – is important if you are using the **Dynamic Data** (p. 103) APIs.

MT Safety:

UNSAFE.

Parameters

name	<<***in***>> *(*p. *433)* The name of the new member. This string must be unique within this type and must not be NULL.
ordinal	<<***in***>> *(*p. *433)* The relative order of the new member in this enum or a custom integer value. The value must be unique within the type.
ex	<<***out***>> *(*p. *433)* If this method fails, this argument will contain information about the failure. Possible values include: • **DDS_BADKIND_USER_EXCEPTION_CODE** (p. 530) • **DDS_BAD_MEMBER_NAME_USER_EXCEPTION_CODE** (p. 531) • **DDS_BAD_MEMBER_ID_USER_EXCEPTION_CODE** (p. 531)

Returns

The zero-based index of the new member relative to any other members that previously existed.

See also

DDS_TypeCode::add_member (p. 1463)

DDS_TypeCode::add_member_ex (p. 1465)

DDS_TypeCodeFactory (p. 1476)

9.214.2.40 add_member_to_union()

```
DDS_UnsignedLong DDS_TypeCode::add_member_to_union (
            const char * name,
             DDS_Long id,
            const  DDS_LongSeq & labels,
            const  DDS_TypeCode * tc,
             DDS_Boolean is_pointer,
             DDS_ExceptionCode_t & ex )
```

Add a new member to a union **DDS_TypeCode** (p. 1428).

This method is applicable to **DDS_TypeCode** (p. 1428) objects representing unions (**DDS_TK_UNION** (p. 89)).

Modifying a **DDS_TypeCode** (p. 1428) – such as by adding a member – is important if you are using the **Dynamic Data** (p. 103) APIs.

MT Safety:

UNSAFE.

Parameters

name	*<<**in**>> (p. 433)* The name of the new member.
id	*<<**in**>> (p. 433)* The member ID. For automatic assignment of the member ID specify **DDS_TYPECODE_MEMBER_ID_INVALID** (p. 83).
labels	*<<**in**>> (p. 433)* A sequence of labels or case values associated with the member .
tc	*<<**in**>> (p. 433)* The type of the new member. You can get or create this **DDS_TypeCode** (p. 1428) with the **DDS_TypeCodeFactory** (p. 1476).
is_pointer	*<<**in**>> (p. 433)* Whether the data member, in its deserialized form, should be stored by pointer as opposed to by value.
ex	*<<**out**>> (p. 433)* If this method fails, this argument will contain information about the failure.

Possible values include:

- **DDS_BADKIND_USER_EXCEPTION_CODE** (p. 530)
- **DDS_BAD_MEMBER_NAME_USER_EXCEPTION_CODE** (p. 531)
- **DDS_BAD_MEMBER_ID_USER_EXCEPTION_CODE** (p. 531)

Returns

The zero-based index of the new member relative to any other members that previously existed.

See also

DDS_TypeCode::add_member (p. 1463)
DDS_TypeCode::add_member_ex (p. 1465)
DDS_TypeCode::add_member_to_enum (p. 1461)
DDS_TypeCode::add_member_to_union (p. 1462)
DDS_TypeCodeFactory (p. 1476)
DDS_TYPECODE_NONKEY_MEMBER (p. 85)
DDS_TYPECODE_KEY_MEMBER (p. 86)
DDS_TYPECODE_NONKEY_REQUIRED_MEMBER (p. 86)

9.214.2.41 add_member()

```
DDS_UnsignedLong DDS_TypeCode::add_member (
            const char * name,
             DDS_Long id,
            const  DDS_TypeCode * tc,
             DDS_Octet member_flags,
             DDS_ExceptionCode_t & ex )
```

Add a new member to this **DDS_TypeCode** (p. 1428).

This method is applicable to **DDS_TypeCode** (p. 1428) objects representing structures (**DDS_TK_STRUCT** (p. 89)), value types (**DDS_TK_VALUE** (p. 89)), and unions (**DDS_TK_UNION** (p. 89)). To add a constant to an enumeration, see **DDS_TypeCode::add_member_to_enum** (p. 1461).

Calling this method clones the type code passed as the `tc` parameter if the type code is not a builtin one. To delete this cloned type code, call **DDS_TypeCodeFactory::delete_tc** (p. 1479).

The ability to modify a **DDS_TypeCode** (p. 1428) – such as by adding a member – is important if you are using the **Dynamic Data** (p. 103) APIs.

Here's a simple code example that adds two fields to a data type, one an integer and another a sequence of integers.

```
// Integer:

myTypeCode->add_member(

    "myFieldName",

    DDS_TYPECODE_MEMBER_ID_INVALID,

    DDS_TheTypeCodeFactory->get_primitive_tc(DDS_TK_LONG),

    // New field is a required non-key member:

    DDS_TYPECODE_NONKEY_REQUIRED_MEMBER);


// Sequence of 10 or fewer integers:

myTypeCode.add_member(

    "myFieldName",

    DDS_TYPECODE_MEMBER_ID_INVALID,

    DDS_TheTypeCodeFactory->create_sequence_tc(

        10,

        DDS_TheTypeCodeFactory->get_primitive_tc(DDS_TK_LONG)),

    // New field is a required non-key member:

    DDS_TYPECODE_NONKEY_REQUIRED_MEMBER);
```

MT Safety:

UNSAFE.

Parameters

name	<<***in***>> *(p. 433)* The name of the new member.
id	<<***in***>> *(p. 433)* Member ID or case value. • For **DDS_TK_STRUCT** (p. 89), and **DDS_TK_VALUE** (p. 89), this parameter is used to provide the member ID. For automatic assignment of the member ID specify **DDS_TYPECODE_MEMBER_ID_INVALID** (p. 83). • For **DDS_TK_UNION** (p. 89), this parameter is used to provide the case value associated with the member. The member ID is assigned automatically. To explicitly assign the member ID, use the API **DDS_TypeCode::add_member_to_union** (p. 1462).
tc	<<***in***>> *(p. 433)* The type of the new member. You can get or create this **DDS_TypeCode** (p. 1428) with the **DDS_TypeCodeFactory** (p. 1476).

Parameters

member_flags	*<<**in**>> (p. 433)* Indicates whether the member is part of the key and whether it is required.
ex	*<<**out**>> (p. 433)* If this method fails, this argument will contain information about the failure. Possible values include: • **DDS_BADKIND_USER_EXCEPTION_CODE** (p. 530) • **DDS_BAD_MEMBER_NAME_USER_EXCEPTION_CODE** (p. 531) • **DDS_BAD_MEMBER_ID_USER_EXCEPTION_CODE** (p. 531)

Returns

The zero-based index of the new member relative to any other members that previously existed.

See also

DDS_TypeCode::add_member_ex (p. 1465)
DDS_TypeCode::add_member_to_enum (p. 1461)
DDS_TypeCodeFactory (p. 1476)
DDS_TYPECODE_NONKEY_MEMBER (p. 85)
DDS_TYPECODE_KEY_MEMBER (p. 86)
DDS_TYPECODE_NONKEY_REQUIRED_MEMBER (p. 86)

9.214.2.42 add_member_ex()

```
DDS_UnsignedLong DDS_TypeCode::add_member_ex (
            const char * name,
             DDS_Long id,
            const  DDS_TypeCode * tc,
             DDS_Octet member_flags,
             DDS_Visibility visibility,
             DDS_Boolean is_pointer,
             DDS_Short bits,
             DDS_Boolean is_must_understand,
             DDS_ExceptionCode_t & ex )
```

Add a new member to this **DDS_TypeCode** (p. 1428).

This method is applicable to **DDS_TypeCode** (p. 1428) objects representing structures (**DDS_TK_STRUCT** (p. 89)), value types (**DDS_TK_VALUE** (p. 89)), and unions (**DDS_TK_UNION** (p. 89)). To add a constant to an enumeration, see **DDS_TypeCode::add_member_to_enum** (p. 1461).

Calling this method clones the type code passed in as the `tc` parameter if the type code is not a builtin one. To delete this cloned type code, call **DDS_TypeCodeFactory::delete_tc** (p. 1479).

The ability to modify a **DDS_TypeCode** (p. 1428) – such as by adding a member – is important if you are using the **Dynamic Data** (p. 103) APIs.

MT Safety:

UNSAFE.

Parameters

name	<<***in***>> *(*p. *433)* The name of the new member.
id	<<***in***>> *(*p. *433)* Member ID or case value. • For **DDS_TK_STRUCT** (p. 89) and **DDS_TK_VALUE** (p. 89), this parameter is used to provide the member ID. • For **DDS_TK_STRUCT** (p. 89) and **DDS_TK_VALUE** (p. 89) only: For automatic assignment of the member ID, specify **DDS_TYPECODE_MEMBER_ID_INVALID** (p. 83). • For **DDS_TK_UNION** (p. 89), this parameter is used to provide the case value associated with the member. The member ID is assigned automatically. To explicitly assign the member ID, use the API **DDS_TypeCode::add_member_to_union** (p. 1462).
tc	<<***in***>> *(*p. *433)* The type of the new member. You can get or create this **DDS_TypeCode** (p. 1428) with the **DDS_TypeCodeFactory** (p. 1476).
member_flags	<<***in***>> *(*p. *433)* Indicates whether the member is part of the key and whether it is required.
visibility	<<***in***>> *(*p. *433)* Whether the new member is public or private. Non-public members are only relevant for types of kind **DDS_TK_VALUE** (p. 89). Possible values include: • **DDS_PRIVATE_MEMBER** (p. 85) • **DDS_PUBLIC_MEMBER** (p. 85)
is_pointer	<<***in***>> *(*p. *433)* Whether the data member, in its deserialized form, should be stored by pointer as opposed to by value.
bits	<<***in***>> *(*p. *433)* The number of bits, if this new member is a bit field, or **DDS_TYPECODE_NOT_BITFIELD** (p. 83).
is_must_understand	<<***in***>> *(*p. *433)* Whether the data member should be marked as "must understand" or not. This value will be ignored and assumed to be true if member_flags indicates that the member is part of the key.
ex	<<***out***>> *(*p. *433)* If this method fails, this argument will contain information about the failure. Possible values include: • **DDS_BADKIND_USER_EXCEPTION_CODE** (p. 530) • **DDS_BAD_MEMBER_NAME_USER_EXCEPTION_CODE** (p. 531) • **DDS_BAD_MEMBER_ID_USER_EXCEPTION_CODE** (p. 531)

Returns

The zero-based index of the new member relative to any other members that previously existed.

See also

DDS_TypeCode::add_member (p. 1463)

DDS_TypeCode::add_member_to_enum (p. 1461)

DDS_TypeCode::add_member_to_union (p. 1462)

DDS_TypeCodeFactory (p. 1476)

DDS_TYPECODE_NONKEY_MEMBER (p. 85)

DDS_TYPECODE_KEY_MEMBER (p. 86)

DDS_TYPECODE_NONKEY_REQUIRED_MEMBER (p. 86)

9.214.2.43 default_value()

```
const DDS_AnnotationParameterValue * DDS_TypeCode::default_value (
            DDS_ExceptionCode_t & ex )
```

Returns the default annotation for this **DDS_TypeCode** (p. 1428).

Precondition

self kind is **DDS_TK_ENUM** (p. 89), or **DDS_TK_ALIAS** (p. 89).

MT Safety:

SAFE

Parameters

ex	<<***out***>> *(p. 433)* Parameter for error indications. The values that can take are: • **DDS_NO_EXCEPTION_CODE** (p. 530) • **DDS_BAD_TYPECODE_SYSTEM_EXCEPTION_CODE** (p. 530) • **DDS_BADKIND_USER_EXCEPTION_CODE** (p. 530)

Returns

the default annotation for this **DDS_TypeCode** (p. 1428)

9.214.2.44 min_value()

```
const DDS_AnnotationParameterValue * DDS_TypeCode::min_value (
            DDS_ExceptionCode_t & ex )
```

Returns the min annotation for this **DDS_TypeCode** (p. 1428).

Precondition

self kind is **DDS_TK_ENUM** (p. 89), or **DDS_TK_ALIAS** (p. 89) of a numerical kind.

MT Safety:

SAFE

Parameters

ex	<<***out***>> *(p. 433)* Parameter for error indications. The values that can take are: • **DDS_NO_EXCEPTION_CODE** (p. 530) • **DDS_BAD_TYPECODE_SYSTEM_EXCEPTION_CODE** (p. 530) • **DDS_BADKIND_USER_EXCEPTION_CODE** (p. 530)

Returns

the min annotation for this **DDS_TypeCode** (p. 1428)

9.214.2.45 max_value()

```
const  DDS_AnnotationParameterValue * DDS_TypeCode::max_value (
             DDS_ExceptionCode_t & ex )
```

Returns the max annotation for this **DDS_TypeCode** (p. 1428).

Precondition

self kind is **DDS_TK_ENUM** (p. 89), or **DDS_TK_ALIAS** (p. 89) of a numerical kind.

MT Safety:

SAFE

Parameters

ex	<<***out***>> *(p. 433)* Parameter for error indications. The values that can take are: • **DDS_NO_EXCEPTION_CODE** (p. 530) • **DDS_BAD_TYPECODE_SYSTEM_EXCEPTION_CODE** (p. 530) • **DDS_BADKIND_USER_EXCEPTION_CODE** (p. 530)

Returns

the max annotation for this **DDS_TypeCode** (p. 1428)

9.214.2.46 unit()

```
const char * DDS_TypeCode::unit (
            DDS_ExceptionCode_t & ex )
```

Returns the unit annotation for this **DDS_TypeCode** (p. 1428).

Precondition

self kind is **DDS_TK_ALIAS** (p. 89).

MT Safety:

SAFE

Parameters

ex	<<***out***>> *(p. 433)* Parameter for error indications. The values that can take are: • **DDS_NO_EXCEPTION_CODE** (p. 530) • **DDS_BAD_TYPECODE_SYSTEM_EXCEPTION_CODE** (p. 530) • **DDS_BADKIND_USER_EXCEPTION_CODE** (p. 530)

Returns

the unit annotation for this **DDS_TypeCode** (p. 1428)

9.214.2.47 member_default_value()

```
const DDS_AnnotationParameterValue * DDS_TypeCode::member_default_value (
        DDS_UnsignedLong index,
        DDS_ExceptionCode_t & ex )
```

Returns the member default annotation for this **DDS_TypeCode** (p. 1428).

Precondition

self kind is **DDS_TK_STRUCT** (p. 89), or **DDS_TK_UNION** (p. 89).

MT Safety:

SAFE

Parameters

index	index of the target member
ex	<<***out***>> *(p. 433)* Parameter for error indications. The values that can take are: • **DDS_NO_EXCEPTION_CODE** (p. 530) • **DDS_BAD_TYPECODE_SYSTEM_EXCEPTION_CODE** (p. 530) • **DDS_BADKIND_USER_EXCEPTION_CODE** (p. 530) • **DDS_BOUNDS_USER_EXCEPTION_CODE** (p. 530)

Returns

the member default annotation for this **DDS_TypeCode** (p. 1428)

9.214.2.48 member_min_value()

```
const DDS_AnnotationParameterValue * DDS_TypeCode::member_min_value (
        DDS_UnsignedLong index,
        DDS_ExceptionCode_t & ex )
```

Returns the member min annotation for this **DDS_TypeCode** (p. 1428).

Precondition

self kind is **DDS_TK_STRUCT** (p. 89), or **DDS_TK_UNION** (p. 89).

member pointed to by index of a numerical kind.

MT Safety:

SAFE

Parameters

index	index of the target member
ex	<<***out***>> *(p. 433)* Parameter for error indications. The values that can take are: • **DDS_NO_EXCEPTION_CODE** (p. 530) • **DDS_BAD_TYPECODE_SYSTEM_EXCEPTION_CODE** (p. 530) • **DDS_BADKIND_USER_EXCEPTION_CODE** (p. 530) • **DDS_BOUNDS_USER_EXCEPTION_CODE** (p. 530)

Returns

the member min annotation for this **DDS_TypeCode** (p. 1428)

9.214.2.49 member_max_value()

```
const  DDS_AnnotationParameterValue * DDS_TypeCode::member_max_value (
             DDS_UnsignedLong index,
             DDS_ExceptionCode_t & ex )
```

Returns the member max annotation for this **DDS_TypeCode** (p. 1428).

Precondition

self kind is **DDS_TK_STRUCT** (p. 89), or **DDS_TK_UNION** (p. 89).

member pointed to by index of a numerical kind.

MT Safety:

SAFE

Parameters

index	index of the target member
ex	<<***out***>> *(p. 433)* Parameter for error indications. The values that can take are: • **DDS_NO_EXCEPTION_CODE** (p. 530) • **DDS_BAD_TYPECODE_SYSTEM_EXCEPTION_CODE** (p. 530) • **DDS_BADKIND_USER_EXCEPTION_CODE** (p. 530) • **DDS_BOUNDS_USER_EXCEPTION_CODE** (p. 530)

Returns

the member max annotation for this **DDS_TypeCode** (p. 1428)

9.214.2.50 member_unit()

```
const char * DDS_TypeCode::member_unit (
              DDS_UnsignedLong index,
              DDS_ExceptionCode_t & ex )
```

Returns the unit annotation for a member of this **DDS_TypeCode** (p. 1428).

Precondition

self kind is **DDS_TK_STRUCT** (p. 89), or **DDS_TK_UNION** (p. 89).

member pointed to by index of a numerical kind.

MT Safety:

SAFE

Parameters

index	index of the target member
ex	<<***out***>> *(p. 433)* Parameter for error indications. The values that can take are: • **DDS_NO_EXCEPTION_CODE** (p. 530) • **DDS_BAD_TYPECODE_SYSTEM_EXCEPTION_CODE** (p. 530) • **DDS_BADKIND_USER_EXCEPTION_CODE** (p. 530) • **DDS_BOUNDS_USER_EXCEPTION_CODE** (p. 530)

Returns

the unit annotation for the specified member of this **DDS_TypeCode** (p. 1428)

9.214.2.51 is_member_must_understand()

```
DDS_Boolean DDS_TypeCode::is_member_must_understand (
            DDS_UnsignedLong index,
            DDS_ExceptionCode_t & ex )
```

Returns the "must understand" property for a member of this **DDS_TypeCode** (p. 1428).

The "must understand" property is typically controlled through annotations, but it may sometimes be constrained by other properties of a member. When invoked on such members, this function will always return a fixed value:

- **DDS_BOOLEAN_TRUE** (p. 513) for members that are part of the key of a struct or value type.
- **DDS_BOOLEAN_FALSE** (p. 513) for any member of a union.

Precondition

self kind is **DDS_TK_STRUCT** (p. 89), **DDS_TK_UNION** (p. 89), or **DDS_TK_VALUE** (p. 89).

member pointed to by index of a numerical kind.

MT Safety:

SAFE

Parameters

index	index of the target member
ex	<<***out***>> *(p. 433)* Parameter for error indications. The values that can take are: • **DDS_NO_EXCEPTION_CODE** (p. 530) • **DDS_BAD_TYPECODE_SYSTEM_EXCEPTION_CODE** (p. 530) • **DDS_BADKIND_USER_EXCEPTION_CODE** (p. 530) • **DDS_BOUNDS_USER_EXCEPTION_CODE** (p. 530)

Returns

the "must understand" property of the selected member of this **DDS_TypeCode** (p. 1428)

9.214.2.52 print_IDL()

```
void DDS_TypeCode::print_IDL (
              DDS_UnsignedLong indent,
              DDS_ExceptionCode_t & ex ) const
```

Prints a **DDS_TypeCode** (p. 1428) in IDL notation.

MT Safety:

SAFE.

Parameters

indent	<<***in***>> *(p. 433)* Indent.
ex	<<***out***>> *(p. 433)* Parameter for error indications. The values that can take are: • **DDS_NO_EXCEPTION_CODE** (p. 530) • **DDS_BAD_TYPECODE_SYSTEM_EXCEPTION_CODE** (p. 530)

9.214.2.53 print()

```
void DDS_TypeCode::print (
             const  DDS_TypeCodePrintFormatProperty & format,
              DDS_ExceptionCode_t & ex ) const
```

Prints a **DDS_TypeCode** (p. 1428).

MT Safety:

SAFE.

Parameters

format	<<***in***>> *(p. 433)* Format.

Parameters

ex	<<***out***>> *(p. 433)* Parameter for error indications. The values that can take are: • **DDS_NO_EXCEPTION_CODE** (p. 530) • **DDS_BAD_TYPECODE_SYSTEM_EXCEPTION_CODE** (p. 530)

9.214.2.54 to_string() [1/2]

```
void DDS_TypeCode::to_string (
             char * str,
              DDS_UnsignedLong & str_size,
              DDS_ExceptionCode_t & ex ) const
```

Get a string representation of this **DDS_TypeCode** (p. 1428) object using the default values for **DDS_TypeCodePrint↩FormatProperty** (p. 1490).

This function obtains a string representation of a **DDS_TypeCode** (p. 1428) object, it creates this string using the default values of **DDS_TypeCodePrintFormatProperty** (p. 1490).

In order to calculate the required length of the string, the user can call this function with the parameter str set to NULL. In this mode, the required length of the string will be stored into the value pointed to by the parameter str_size.

If the parameter str is not equal to NULL, the string representation of the **DDS_TypeCode** (p. 1428) will be stored into the parameter str. In this mode, if the size of the string is insufficient to hold the result, the parameter ∗ex will be set to **DDS_BAD_PARAM_SYSTEM_EXCEPTION_CODE** (p. 530) and the parameter str_size will be updated to hold the required size of the string. All string lengths include the trailing NUL byte.

Parameters

str	<<***out***>> *(p. 433)* The char buffer that will be used to store the string representation of the **DDS_TypeCode** (p. 1428). If NULL then this function will store the required length of the buffer into the value pointed to by the str_size parameter.
str_size	<<***inout***>> *(p. 433)* Cannot be NULL. The length of the char buffer. If the supplied buffer is NULL or insufficiently long, the value pointed to by this parameter will be updated to contain the required length.
ex	<<***out***>> *(p. 433)* Parameter used for error indications.

See also

DDS_TypeCode::print_IDL (p. 1474)

DDS_TypeCode::print (p. 1474)

DDS_TypeCode::to_string (p. 1475)

9.214.2.55 to_string() [2/2]

```
void DDS_TypeCode::to_string (
            char * str,
             DDS_UnsignedLong & str_size,
            const  DDS_TypeCodePrintFormatProperty & format,
             DDS_ExceptionCode_t & ex ) const
```

Get a string representation of this **DDS_TypeCode** (p. 1428) object using the format described by **DDS_TypeCode↩ PrintFormatProperty** (p. 1490).

This function behaves in the same way as **DDS_TypeCode::to_string** (p. 1475) but allows the user to specify the format of the string using **DDS_TypeCodePrintFormatProperty** (p. 1490).

Parameters

str	<<***out***>> *(*p. *433)* The char buffer which will hold the string representation of the **DDS_TypeCode** (p. 1428).
str_size	<<***inout***>> *(*p. *433)* Cannot be NULL. The length of the char buffer pointed to by the str parameter. If the str parameter is NULL, or too small, this parameter will be updated to hold the required length of the buffer.
format	<<***in***>> *(*p. *433)* The **DDS_TypeCodePrintFormatProperty** (p. 1490) used to define the format of the string representation of the **DDS_TypeCode** (p. 1428).
ex	<<***out***>> *(*p. *433)* Parameter used for error indications.

See also

DDS_TypeCode::print_IDL (p. 1474)

DDS_TypeCode::print (p. 1474)

DDS_TypeCode::to_string (p. 1475)

9.215 DDS_TypeCodeFactory Struct Reference

A singleton factory for creating, copying, and deleting data type definitions dynamically.

Public Member Functions

- **DDS_TypeCode** ∗ **clone_tc** (const **DDS_TypeCode** ∗tc, **DDS_ExceptionCode_t** &ex)

  *Creates and returns a copy of the input **DDS_TypeCode** (*p. *1428).*
- void **delete_tc** (**DDS_TypeCode** ∗tc, **DDS_ExceptionCode_t** &ex)

  *Deletes the input **DDS_TypeCode** (*p. *1428).*
- const **DDS_TypeCode** ∗ **get_primitive_tc** (**DDS_TCKind** tc_kind)

  *Get the **DDS_TypeCode** (*p. *1428) for a primitive type (integers, floating point values, etc.) identified by the given **DDS↩ _TCKind** (*p. *89).*
- **DDS_TypeCode** ∗ **create_struct_tc** (const char ∗name, const **DDS_StructMemberSeq** &members, **DDS_↩ ExceptionCode_t** &ex)

*Constructs a **DDS_TK_STRUCT** (p. 89) **DDS_TypeCode** (p. 1428).*

- **DDS_TypeCode** * **create_struct_tc** (const char *name, **DDS_ExtensibilityKind** extensibility_kind, const **DDS_StructMemberSeq** &members, **DDS_ExceptionCode_t** &ex)

  *Constructs a **DDS_TK_STRUCT** (p. 89) **DDS_TypeCode** (p. 1428).*
- **DDS_TypeCode** * **create_value_tc** (const char *name, **DDS_ValueModifier** type_modifier, const **DDS_TypeCode** *concrete_base, const **DDS_ValueMemberSeq** &members, **DDS_ExceptionCode_t** &ex)

  *Constructs a **DDS_TK_VALUE** (p. 89) **DDS_TypeCode** (p. 1428).*
- **DDS_TypeCode** * **create_value_tc** (const char *name, **DDS_ExtensibilityKind** extensibility_kind, **DDS_ValueModifier** type_modifier, const **DDS_TypeCode** *concrete_base, const **DDS_ValueMemberSeq** &members, **DDS_ExceptionCode_t** &ex)

  *Constructs a **DDS_TK_VALUE** (p. 89) **DDS_TypeCode** (p. 1428).*
- **DDS_TypeCode** * **create_union_tc** (const char *name, const **DDS_TypeCode** *discriminator_type, **DDS_Long** default_index, const **DDS_UnionMemberSeq** &members, **DDS_ExceptionCode_t** &ex)

  *Constructs a **DDS_TK_UNION** (p. 89) **DDS_TypeCode** (p. 1428).*
- **DDS_TypeCode** * **create_union_tc** (const char *name, **DDS_ExtensibilityKind** extensibility_kind, const **DDS_TypeCode** *discriminator_type, **DDS_Long** default_index, const **DDS_UnionMemberSeq** &members, **DDS_ExceptionCode_t** &ex)

  *Constructs a **DDS_TK_UNION** (p. 89) **DDS_TypeCode** (p. 1428).*
- **DDS_TypeCode** * **create_enum_tc** (const char *name, const **DDS_EnumMemberSeq** &members, **DDS_ExceptionCode_t** &ex)

  *Constructs a **DDS_TK_ENUM** (p. 89) **DDS_TypeCode** (p. 1428).*
- **DDS_TypeCode** * **create_enum_tc** (const char *name, **DDS_ExtensibilityKind** extensibility_kind, const **DDS_EnumMemberSeq** &members, **DDS_ExceptionCode_t** &ex)

  *Constructs a **DDS_TK_ENUM** (p. 89) **DDS_TypeCode** (p. 1428).*
- **DDS_TypeCode** * **create_alias_tc** (const char *name, const **DDS_TypeCode** *original_type, **DDS_Boolean** is_pointer, **DDS_ExceptionCode_t** &ex)

  *Constructs a **DDS_TK_ALIAS** (p. 89) (typedef) **DDS_TypeCode** (p. 1428).*
- **DDS_TypeCode** * **create_string_tc** (**DDS_UnsignedLong** bound, **DDS_ExceptionCode_t** &ex)

  *Constructs a **DDS_TK_STRING** (p. 89) **DDS_TypeCode** (p. 1428).*
- **DDS_TypeCode** * **create_wstring_tc** (**DDS_UnsignedLong** bound, **DDS_ExceptionCode_t** &ex)

  *Constructs a **DDS_TK_WSTRING** (p. 89) **DDS_TypeCode** (p. 1428).*
- **DDS_TypeCode** * **create_sequence_tc** (**DDS_UnsignedLong** bound, const **DDS_TypeCode** *element_type, **DDS_ExceptionCode_t** &ex)

  *Constructs a **DDS_TK_SEQUENCE** (p. 89) **DDS_TypeCode** (p. 1428).*
- **DDS_TypeCode** * **create_array_tc** (const **DDS_UnsignedLongSeq** &dimensions, const **DDS_TypeCode** *element_type, **DDS_ExceptionCode_t** &ex)

  *Constructs a **DDS_TK_ARRAY** (p. 89) **DDS_TypeCode** (p. 1428).*
- **DDS_TypeCode** * **create_array_tc** (**DDS_UnsignedLong** length, const **DDS_TypeCode** *element_type, **DDS_ExceptionCode_t** &ex)

  *Constructs a **DDS_TK_ARRAY** (p. 89) **DDS_TypeCode** (p. 1428) for a single-dimensional array.*

Static Public Member Functions

- static **DDS_TypeCodeFactory** * **get_instance** ()

  Gets the singleton instance of this class.

9.215.1 Detailed Description

A singleton factory for creating, copying, and deleting data type definitions dynamically.

You can access the singleton with the **DDS_TypeCodeFactory::get_instance** (p. 1479) method.

If you want to publish and subscribe to data of types that are not known to you at system design time, this class will be your starting point. After creating a data type definition with this class, you will modify that definition using the **DDS_TypeCode** (p. 1428) class and then register it with the **Dynamic Data** (p. 103) API.

The methods of this class fall into several categories:

Getting definitions for primitive types:
Type definitions for primitive types (e.g. integers, floating point values, etc.) are pre-defined; your application only needs to *get* them, not *create* them.

- **DDS_TypeCodeFactory::get_primitive_tc** (p. 1480)

Creating definitions for strings, arrays, and sequences:
Type definitions for strings, arrays, and sequences (i.e. variables-size lists) must be created as you need them, because the type definition includes the maximum length of those containers.

- **DDS_TypeCodeFactory::create_string_tc** (p. 1487)
- **DDS_TypeCodeFactory::create_wstring_tc** (p. 1487)
- **DDS_TypeCodeFactory::create_array_tc** (p. 1489)
- **DDS_TypeCodeFactory::create_array_tc** (p. 1489)
- **DDS_TypeCodeFactory::create_sequence_tc** (p. 1488)

Creating definitions for structured types:
Structured types include structures, value types, and unions.

- **DDS_TypeCodeFactory::create_struct_tc** (p. 1480)
- **DDS_TypeCodeFactory::create_value_tc** (p. 1482)
- **DDS_TypeCodeFactory::create_union_tc** (p. 1483)

Creating definitions for other types:
The type system also supports enumerations and aliases (i.e. `typedef`s in C and C++).

- **DDS_TypeCodeFactory::create_enum_tc** (p. 1485)
- **DDS_TypeCodeFactory::create_alias_tc** (p. 1486)

Deleting type definitions:
When you're finished using a type definition, you should delete it. (*Note* that you only need to delete a **DDS_TypeCode** (p. 1428) that you *created*; if you got the object from **DDS_TypeCodeFactory::get_primitive_tc** (p. 1480), you must *not* delete it.)

- **DDS_TypeCodeFactory::delete_tc** (p. 1479)

Copying type definitions:
You can also create deep copies of type definitions:

- **DDS_TypeCodeFactory::clone_tc** (p. 1479)

9.215.2 Member Function Documentation

9.215.2.1 get_instance()

```
static  DDS_TypeCodeFactory * DDS_TypeCodeFactory::get_instance ( )  [static]
```

Gets the singleton instance of this class.

Returns

The **DDS_TypeCodeFactory** (p. 1476) instance if no errors. Otherwise, NULL.

9.215.2.2 clone_tc()

```
DDS_TypeCode * DDS_TypeCodeFactory::clone_tc (
            const  DDS_TypeCode * tc,
             DDS_ExceptionCode_t & ex )
```

Creates and returns a copy of the input **DDS_TypeCode** (p. 1428).

Parameters

tc	<<***in***>> *(p. 433)* Type code that will be copied. Cannot be NULL.
ex	<<***out***>> *(p. 433)* Parameter for error indications. The values that can take are: • **DDS_NO_EXCEPTION_CODE** (p. 530) • **DDS_BAD_PARAM_SYSTEM_EXCEPTION_CODE** (p. 530) • **DDS_NO_MEMORY_SYSTEM_EXCEPTION_CODE** (p. 530) • **DDS_BAD_TYPECODE_SYSTEM_EXCEPTION_CODE** (p. 530)

Returns

A clone of tc.

9.215.2.3 delete_tc()

```
void DDS_TypeCodeFactory::delete_tc (
                DDS_TypeCode * tc,
                DDS_ExceptionCode_t & ex )
```

Deletes the input **DDS_TypeCode** (p. 1428).

All the type codes created through the **DDS_TypeCodeFactory** (p. 1476) must be deleted using this method.

Parameters

tc	<<***inout***>> *(p. 433)* Type code that will be deleted. Cannot be NULL.
ex	<<***out***>> *(p. 433)* Parameter for error indications. The values that can take are: • **DDS_NO_EXCEPTION_CODE** (p. 530) • **DDS_BAD_PARAM_SYSTEM_EXCEPTION_CODE** (p. 530)

9.215.2.4 get_primitive_tc()

```
const  DDS_TypeCode * DDS_TypeCodeFactory::get_primitive_tc (
                DDS_TCKind tc_kind )
```

Get the **DDS_TypeCode** (p. 1428) for a primitive type (integers, floating point values, etc.) identified by the given **DDS_TCKind** (p. 89).

This method is equivalent to, and replaces, the `DDS_g_tc_*` constants.

See also

DDS_g_tc_long (p. 91)
DDS_g_tc_ulong (p. 91)
DDS_g_tc_short (p. 90)
DDS_g_tc_ushort (p. 91)
DDS_g_tc_float (p. 92)
DDS_g_tc_double (p. 92)
DDS_g_tc_longdouble (p. 95)
DDS_g_tc_octet (p. 93)
DDS_g_tc_boolean (p. 92)
DDS_g_tc_char (p. 93)
DDS_g_tc_wchar (p. 95)

9.215.2.5 create_struct_tc() [1/2]

```
DDS_TypeCode * DDS_TypeCodeFactory::create_struct_tc (
            const char * name,
            const  DDS_StructMemberSeq & members,
             DDS_ExceptionCode_t & ex )
```

Constructs a **DDS_TK_STRUCT** (p. 89) **DDS_TypeCode** (p. 1428).

Parameters

name	<<***in***>> *(p. 433)* Name of the struct type. Cannot be NULL.
members	<<***in***>> *(p. 433)* Initial members of the structure. This list may be empty (that is, **FooSeq::length()** (p. 1973) may return zero). If the list is not empty, the elements must describe valid struct members. (For example, the names must be unique within the type.)
ex	<<***out***>> *(p. 433)* Parameter for error indications. The values that can take are: • **DDS_NO_EXCEPTION_CODE** (p. 530) • **DDS_BAD_PARAM_SYSTEM_EXCEPTION_CODE** (p. 530) • **DDS_NO_MEMORY_SYSTEM_EXCEPTION_CODE** (p. 530) • **DDS_BAD_TYPECODE_SYSTEM_EXCEPTION_CODE** (p. 530)

Returns

A newly-created **DDS_TypeCode** (p. 1428) object describing a struct.

9.215.2.6 create_struct_tc() [2/2]

```
DDS_TypeCode * DDS_TypeCodeFactory::create_struct_tc (
            const char * name,
             DDS_ExtensibilityKind extensibility_kind,
            const  DDS_StructMemberSeq & members,
             DDS_ExceptionCode_t & ex )
```

Constructs a **DDS_TK_STRUCT** (p. 89) **DDS_TypeCode** (p. 1428).

Parameters

name	<<***in***>> *(p. 433)* Name of the struct type. Cannot be NULL.
extensibility_kind	<<***in***>> *(p. 433)* Type extensibility.
members	<<***in***>> *(p. 433)* Initial members of the structure. This list may be empty (that is, **FooSeq::length()** (p. 1973) may return zero). If the list is not empty, the elements must describe valid struct members. (For example, the names must be unique within the type.)

Parameters

ex	<<***out***>> *(p. 433)* Parameter for error indications. The values that can take are: • **DDS_NO_EXCEPTION_CODE** (p. 530) • **DDS_BAD_PARAM_SYSTEM_EXCEPTION_CODE** (p. 530) • **DDS_NO_MEMORY_SYSTEM_EXCEPTION_CODE** (p. 530) • **DDS_BAD_TYPECODE_SYSTEM_EXCEPTION_CODE** (p. 530)

Returns

A newly-created **DDS_TypeCode** (p. 1428) object describing a struct.

9.215.2.7 create_value_tc() [1/2]

```
DDS_TypeCode * DDS_TypeCodeFactory::create_value_tc (
            const char * name,
             DDS_ValueModifier type_modifier,
            const  DDS_TypeCode * concrete_base,
            const  DDS_ValueMemberSeq & members,
             DDS_ExceptionCode_t & ex )
```

Constructs a **DDS_TK_VALUE** (p. 89) **DDS_TypeCode** (p. 1428).

Parameters

name	<<***in***>> *(p. 433)* Name of the value type. Cannot be NULL.
type_modifier	<<***in***>> *(p. 433)* One of the value type modifier constants: **DDS_VM_NONE** (p. 84), **DDS_VM_CUSTOM** (p. 84), **DDS_VM_ABSTRACT** (p. 84) or **DDS_VM_TRUNCATABLE** (p. 84).
concrete_base	<<***in***>> *(p. 433)* **DDS_TypeCode** (p. 1428) object describing the concrete valuetype base. It may be NULL if the valuetype does not have a concrete base.
members	<<***in***>> *(p. 433)* Initial members of the value type. This list may be empty. If the list is not empty, the elements must describe valid value type members. (For example, the names must be unique within the type.)
ex	<<***out***>> *(p. 433)* Parameter for error indications. The values that can take are: • **DDS_NO_EXCEPTION_CODE** (p. 530) • **DDS_BAD_PARAM_SYSTEM_EXCEPTION_CODE** (p. 530) • **DDS_NO_MEMORY_SYSTEM_EXCEPTION_CODE** (p. 530) • **DDS_BAD_TYPECODE_SYSTEM_EXCEPTION_CODE** (p. 530)

Returns

A newly-created **DDS_TypeCode** (p. 1428) object describing a value.

9.215.2.8 create_value_tc() [2/2]

```
DDS_TypeCode * DDS_TypeCodeFactory::create_value_tc (
            const char * name,
             DDS_ExtensibilityKind extensibility_kind,
             DDS_ValueModifier type_modifier,
            const  DDS_TypeCode * concrete_base,
            const  DDS_ValueMemberSeq & members,
             DDS_ExceptionCode_t & ex )
```

Constructs a **DDS_TK_VALUE** (p. 89) **DDS_TypeCode** (p. 1428).

Parameters

name	<<***in***>> *(p. 433)* Name of the value type. Cannot be NULL.
extensibility_kind	<<***in***>> *(p. 433)* Type extensibility.
type_modifier	<<***in***>> *(p. 433)* One of the value type modifier constants: **DDS_VM_NONE** (p. 84), **DDS_VM_CUSTOM** (p. 84), **DDS_VM_ABSTRACT** (p. 84) or **DDS_VM_TRUNCATABLE** (p. 84).
concrete_base	<<***in***>> *(p. 433)* **DDS_TypeCode** (p. 1428) object describing the concrete valuetype base. It may be NULL if the valuetype does not have a concrete base.
members	<<***in***>> *(p. 433)* Initial members of the value type. This list may be empty. If the list is not empty, the elements must describe valid value type members. (For example, the names must be unique within the type.)
ex	<<***out***>> *(p. 433)* Parameter for error indications. The values that can take are: • **DDS_NO_EXCEPTION_CODE** (p. 530) • **DDS_BAD_PARAM_SYSTEM_EXCEPTION_CODE** (p. 530) • **DDS_NO_MEMORY_SYSTEM_EXCEPTION_CODE** (p. 530) • **DDS_BAD_TYPECODE_SYSTEM_EXCEPTION_CODE** (p. 530)

Returns

A newly-created **DDS_TypeCode** (p. 1428) object describing a value.

9.215.2.9 create_union_tc() [1/2]

```
DDS_TypeCode * DDS_TypeCodeFactory::create_union_tc (
            const char * name,
```

```
const  DDS_TypeCode * discriminator_type,
 DDS_Long default_index,
const  DDS_UnionMemberSeq & members,
 DDS_ExceptionCode_t & ex )
```

Constructs a **DDS_TK_UNION** (p. 89) **DDS_TypeCode** (p. 1428).

Parameters

name	<<***in***>> *(p. 433)* Name of the union type. Cannot be NULL.
discriminator_type	<<***in***>> *(p. 433)* Discriminator Type Code. Cannot be NULL.
default_index	<<***in***>> *(p. 433)* Index of the default member, or -1 if there is no default member.
members	<<***in***>> *(p. 433)* Initial members of the union. This list may be empty. If the list is not empty, the elements must describe valid struct members. (For example, the names must be unique within the type.)
ex	<<***out***>> *(p. 433)* Parameter for error indications. The values that can take are: • **DDS_NO_EXCEPTION_CODE** (p. 530) • **DDS_BAD_PARAM_SYSTEM_EXCEPTION_CODE** (p. 530) • **DDS_NO_MEMORY_SYSTEM_EXCEPTION_CODE** (p. 530) • **DDS_BAD_TYPECODE_SYSTEM_EXCEPTION_CODE** (p. 530)

Returns

A newly-created **DDS_TypeCode** (p. 1428) object describing a union.

9.215.2.10 create_union_tc() [2/2]

```
DDS_TypeCode * DDS_TypeCodeFactory::create_union_tc (
            const char * name,
             DDS_ExtensibilityKind extensibility_kind,
            const  DDS_TypeCode * discriminator_type,
             DDS_Long default_index,
            const  DDS_UnionMemberSeq & members,
             DDS_ExceptionCode_t & ex )
```

Constructs a **DDS_TK_UNION** (p. 89) **DDS_TypeCode** (p. 1428).

Parameters

name	<<***in***>> *(p. 433)* Name of the union type. Cannot be NULL.
extensibility_kind	<<***in***>> *(p. 433)* Type extensibility.
discriminator_type	<<***in***>> *(p. 433)* Discriminator Type Code. Cannot be NULL.
default_index	<<***in***>> *(p. 433)* Index of the default member, or -1 if there is no default member.

Parameters

members	<<***in***>> *(p. 433)* Initial members of the union. This list may be empty. If the list is not empty, the elements must describe valid struct members. (For example, the names must be unique within the type.)
ex	<<***out***>> *(p. 433)* Parameter for error indications. The values that can take are: • **DDS_NO_EXCEPTION_CODE** (p. 530) • **DDS_BAD_PARAM_SYSTEM_EXCEPTION_CODE** (p. 530) • **DDS_NO_MEMORY_SYSTEM_EXCEPTION_CODE** (p. 530) • **DDS_BAD_TYPECODE_SYSTEM_EXCEPTION_CODE** (p. 530)

Returns

A newly-created **DDS_TypeCode** (p. 1428) object describing a union.

9.215.2.11 create_enum_tc() [1/2]

```
DDS_TypeCode * DDS_TypeCodeFactory::create_enum_tc (
            const char * name,
            const  DDS_EnumMemberSeq & members,
             DDS_ExceptionCode_t & ex )
```

Constructs a **DDS_TK_ENUM** (p. 89) **DDS_TypeCode** (p. 1428).

Parameters

name	<<***in***>> *(p. 433)* Name of the enum type. Cannot be NULL.
members	<<***in***>> *(p. 433)* Initial members of the enumeration. All members must have non-NULL names, and both names and ordinal values must be unique within the type. Note that it is also possible to add members later with **DDS_TypeCode::add_member_to_enum** (p. 1461).
ex	<<***out***>> *(p. 433)* Parameter for error indications. The values that can take are: • **DDS_NO_EXCEPTION_CODE** (p. 530) • **DDS_BAD_PARAM_SYSTEM_EXCEPTION_CODE** (p. 530) • **DDS_NO_MEMORY_SYSTEM_EXCEPTION_CODE** (p. 530)

Returns

A newly-created **DDS_TypeCode** (p. 1428) object describing an enumeration.

9.215.2.12 create_enum_tc() [2/2]

```
DDS_TypeCode * DDS_TypeCodeFactory::create_enum_tc (
            const char * name,
             DDS_ExtensibilityKind extensibility_kind,
            const  DDS_EnumMemberSeq & members,
             DDS_ExceptionCode_t & ex )
```

Constructs a **DDS_TK_ENUM** (p. 89) **DDS_TypeCode** (p. 1428).

Parameters

name	<<***in***>> *(p. 433)* Name of the enum type. Cannot be NULL.
extensibility_kind	<<***in***>> *(p. 433)* Type extensibility.
members	<<***in***>> *(p. 433)* Initial members of the enumeration. All members must have non-NULL names, and both names and ordinal values must be unique within the type. Note that it is also possible to add members later with **DDS_TypeCode::add_member_to_enum** (p. 1461).
ex	<<***out***>> *(p. 433)* Parameter for error indications. The values that can take are: • **DDS_NO_EXCEPTION_CODE** (p. 530) • **DDS_BAD_PARAM_SYSTEM_EXCEPTION_CODE** (p. 530) • **DDS_NO_MEMORY_SYSTEM_EXCEPTION_CODE** (p. 530)

Returns

A newly-created **DDS_TypeCode** (p. 1428) object describing an enumeration.

9.215.2.13 create_alias_tc()

```
DDS_TypeCode * DDS_TypeCodeFactory::create_alias_tc (
            const char * name,
            const  DDS_TypeCode * original_type,
             DDS_Boolean is_pointer,
             DDS_ExceptionCode_t & ex )
```

Constructs a **DDS_TK_ALIAS** (p. 89) (typedef) **DDS_TypeCode** (p. 1428).

Parameters

name	<<***in***>> *(p. 433)* Name of the alias. Cannot be NULL.
original_type	<<***in***>> *(p. 433)* Aliased type code. Cannot be NULL.

Parameters

is_pointer	<<***in***>> *(*p. *433)* Indicates if the alias is a pointer to the aliased type code.
ex	<<***out***>> *(*p. *433)* Parameter for error indications. The values that can take are: • **DDS_NO_EXCEPTION_CODE** (p. 530) • **DDS_BAD_PARAM_SYSTEM_EXCEPTION_CODE** (p. 530) • **DDS_NO_MEMORY_SYSTEM_EXCEPTION_CODE** (p. 530) • **DDS_BAD_TYPECODE_SYSTEM_EXCEPTION_CODE** (p. 530)

Returns

A newly-created **DDS_TypeCode** (p. 1428) object describing an alias.

9.215.2.14 create_string_tc()

```
DDS_TypeCode * DDS_TypeCodeFactory::create_string_tc (
            DDS_UnsignedLong bound,
            DDS_ExceptionCode_t & ex )
```

Constructs a **DDS_TK_STRING** (p. 89) **DDS_TypeCode** (p. 1428).

Parameters

bound	<<***in***>> *(*p. *433)* Maximum length of the string.
ex	<<***out***>> *(*p. *433)* Parameter for error indications. The values that can take are: • **DDS_NO_EXCEPTION_CODE** (p. 530) • **DDS_BAD_PARAM_SYSTEM_EXCEPTION_CODE** (p. 530) • **DDS_NO_MEMORY_SYSTEM_EXCEPTION_CODE** (p. 530)

Returns

A newly-created **DDS_TypeCode** (p. 1428) object describing a string.

9.215.2.15 create_wstring_tc()

```
DDS_TypeCode * DDS_TypeCodeFactory::create_wstring_tc (
            DDS_UnsignedLong bound,
            DDS_ExceptionCode_t & ex )
```

Constructs a **DDS_TK_WSTRING** (p. 89) **DDS_TypeCode** (p. 1428).

Parameters

bound	<<***in***>> *(p. 433)* Maximum length of the wide string.
ex	<<***out***>> *(p. 433)* Parameter for error indications. The values that can take are: • **DDS_NO_EXCEPTION_CODE** (p. 530) • **DDS_BAD_PARAM_SYSTEM_EXCEPTION_CODE** (p. 530) • **DDS_NO_MEMORY_SYSTEM_EXCEPTION_CODE** (p. 530)

Returns

A newly-created **DDS_TypeCode** (p. 1428) object describing a wide string.

9.215.2.16 create_sequence_tc()

```
DDS_TypeCode * DDS_TypeCodeFactory::create_sequence_tc (
            DDS_UnsignedLong bound,
           const  DDS_TypeCode * element_type,
            DDS_ExceptionCode_t & ex )
```

Constructs a **DDS_TK_SEQUENCE** (p. 89) **DDS_TypeCode** (p. 1428).

Parameters

bound	<<***in***>> *(p. 433)* The bound for the sequence (> 0).
element_type	<<***in***>> *(p. 433)* **DDS_TypeCode** (p. 1428) object describing the sequence elements.
ex	<<***out***>> *(p. 433)* Parameter for error indications. The values that can take are: • **DDS_NO_EXCEPTION_CODE** (p. 530) • **DDS_BAD_PARAM_SYSTEM_EXCEPTION_CODE** (p. 530) • **DDS_NO_MEMORY_SYSTEM_EXCEPTION_CODE** (p. 530) • **DDS_BAD_TYPECODE_SYSTEM_EXCEPTION_CODE** (p. 530)

Returns

A newly-created **DDS_TypeCode** (p. 1428) object describing a sequence.

9.215.2.17 create_array_tc() [1/2]

```
DDS_TypeCode * DDS_TypeCodeFactory::create_array_tc (
            const  DDS_UnsignedLongSeq & dimensions,
            const  DDS_TypeCode * element_type,
             DDS_ExceptionCode_t & ex )
```

Constructs a **DDS_TK_ARRAY** (p. 89) **DDS_TypeCode** (p. 1428).

Parameters

dimensions	<<***in***>> *(*p. *433)* Dimensions of the array. Each dimension has to be greater than 0.
element_type	<<***in***>> *(*p. *433)* **DDS_TypeCode** (p. 1428) describing the array elements. Cannot be NULL.
ex	<<***out***>> *(*p. *433)* Parameter for error indications. The values that can take are: • **DDS_NO_EXCEPTION_CODE** (p. 530) • **DDS_BAD_PARAM_SYSTEM_EXCEPTION_CODE** (p. 530) • **DDS_NO_MEMORY_SYSTEM_EXCEPTION_CODE** (p. 530) • **DDS_BAD_TYPECODE_SYSTEM_EXCEPTION_CODE** (p. 530)

Returns

A newly-created **DDS_TypeCode** (p. 1428) object describing a sequence.

9.215.2.18 create_array_tc() [2/2]

```
DDS_TypeCode * DDS_TypeCodeFactory::create_array_tc (
             DDS_UnsignedLong length,
            const  DDS_TypeCode * element_type,
             DDS_ExceptionCode_t & ex )
```

Constructs a **DDS_TK_ARRAY** (p. 89) **DDS_TypeCode** (p. 1428) for a single-dimensional array.

Parameters

length	<<***in***>> *(*p. *433)* Length of the single-dimensional array.

Parameters

element_type	<<***in***>> *(*p. *433)* **DDS_TypeCode** (p. 1428) describing the array elements. Cannot be NULL.
ex	<<***out***>> *(*p. *433)* Parameter for error indications. The values that can take are: • **DDS_NO_EXCEPTION_CODE** (p. 530) • **DDS_BAD_PARAM_SYSTEM_EXCEPTION_CODE** (p. 530) • **DDS_NO_MEMORY_SYSTEM_EXCEPTION_CODE** (p. 530) • **DDS_BAD_TYPECODE_SYSTEM_EXCEPTION_CODE** (p. 530)

Returns

A newly-created **DDS_TypeCode** (p. 1428) object describing a sequence.

9.216 DDS_TypeCodePrintFormatProperty Struct Reference

A collection of attributes used to configure how a TypeCode appears when converted to a string.

Public Attributes

- **DDS_Long indent**

  Configures how much indent should be added to the string representation of a ***DDS_TypeCode*** *(*p. *1428).*
- **DDS_Boolean print_ordinals**

  Configures whether or not to print the ordinal value of each enumerator within a ***DDS_TypeCode*** *(*p. *1428).*
- **DDS_TypeCodePrintFormatKind print_kind**

  Configures whether the type should be printed in XML or IDL format.
- **DDS_Boolean print_complete_type**

  Configures whether or not to print the complete type.

9.216.1 Detailed Description

A collection of attributes used to configure how a TypeCode appears when converted to a string.

To ensure that new objects are initialized to a known value, assign them with the static initializer **DDS_TypeCode_↩ PrintFormat_INITIALIZER** (p. 87).

9.216.2 Member Data Documentation

9.216.2.1 indent

DDS_Long DDS_TypeCodePrintFormatProperty::indent

Configures how much indent should be added to the string representation of a **DDS_TypeCode** (p. 1428).

Configures how much additional indent is applied when converting a TypeCode to a string. This value acts as a total offset on the string, increasing the indent applied to all elements by the same amount. With an indent of 0, a string representation of a TypeCode may appear as:

```
struct myType {
    long x;
};
```

Using an indent of 1, the same TypeCode would be printed as:

```
    struct myType {
        long x;
    };
```

I.e., the entire structure is indented.

9.216.2.2 print_ordinals

DDS_Boolean DDS_TypeCodePrintFormatProperty::print_ordinals

Configures whether or not to print the ordinal value of each enumerator within a **DDS_TypeCode** (p. 1428).

When set to true, the ordinal value of each enumerator within an enum will be printed, otherwise only non-default ordinals are printed. Take for example the following enum:

```
enum myEnum {
    RED,
    GREEN = 3,
    BLUE,
};
```

When print_ordinals is set to false it would be printed as:

```
enum myEnum {
    RED,
    GREEN = 3,
    BLUE,
};
```

But with print_ordinals set to true it would be printed as:

```
enum myEnum {
    RED = 0,
    GREEN = 3,
    BLUE = 4,
};
```

9.216.2.3 print_kind

DDS_TypeCodePrintFormatKind DDS_TypeCodePrintFormatProperty::print_kind

Configures whether the type should be printed in XML or IDL format.

When print_kind is DDS_TYPE_CODE_PRINT_KIND_IDL, the type will be printed in IDL format. For example:

```
struct Foo {
    float32 bar;
};
```

When print_kind is DDS_TYPE_CODE_PRINT_KIND_XML, the type will be printed in XML format. For example:

```
<struct name="Foo">
    <member name="bar" type="float32"/>
</struct>
```

9.216.2.4 print_complete_type

DDS_Boolean DDS_TypeCodePrintFormatProperty::print_complete_type

Configures whether or not to print the complete type.

When print_complete_type is true, the complete type will be printed. When print_complete_type is false, only the top level will be printed.

Take for example the following types:

```
struct Foo {

    float32 member;

};

struct Bar {

    Foo foo;

};
```

When print_complete_type is false, this is printed as:

```
struct Bar {

    Foo foo;

};
```

When print_complete_type is true, this is printed as:

```
struct Foo {

    float32 member;

};

struct Bar {

    Foo foo;

};
```

9.217 DDS_TypeConsistencyEnforcementQosPolicy Struct Reference

<<***spec***>> *(*p. *209)* Defines the rules for determining whether the type used to publish a given topic is consistent with that used to subscribe to it.

Public Attributes

- **DDS_TypeConsistencyKind kind**

  <<***spec***>> *(*p. *209) Type consistency kind.*
- **DDS_Boolean ignore_sequence_bounds**

  <<***spec***>> *(*p. *209) Controls whether sequence bounds are taken into consideration for type assignability.*
- **DDS_Boolean ignore_string_bounds**

  <<***spec***>> *(*p. *209) Controls whether string bounds are taken into consideration for type assignability.*
- **DDS_Boolean ignore_member_names**

  <<***spec***>> *(*p. *209) Controls whether member names are taken into consideration for type assignability.*
- **DDS_Boolean prevent_type_widening**

  <<***spec***>> *(*p. *209) Controls whether type widening is allowed.*
- **DDS_Boolean force_type_validation**

  <<***spec***>> *(*p. *209) Controls whether type information must be available in order to complete matching between a* ***DDSDataWriter*** *(*p. *1589) and a* ***DDSDataReader*** *(*p. *1556).*
- **DDS_Boolean ignore_enum_literal_names**

  <<***spec***>> *(*p. *209) Controls whether enumeration constant names are taken into consideration for type assignability.*

9.217.1 Detailed Description

<<***spec***>> *(p. 209)* Defines the rules for determining whether the type used to publish a given topic is consistent with that used to subscribe to it.

This policy defines a type consistency kind, which allows applications to select from among a set of predetermined behaviors. The following consistency kinds are specified: **DDS_DISALLOW_TYPE_COERCION** (p. 651), **DDS_ALLOW_TYPE_COERCION** (p. 651) and **DDS_AUTO_TYPE_COERCION** (p. 651).

The type-consistency-enforcement rules consist of two steps:

Step 1. If both the DataWriter and DataReader specify a TypeObject, it is considered first. If the DataReader allows type coercion, then its type must be assignable from the DataWriter's type, taking into account the values of prevent_type_widening, ignore_sequence_bounds, ignore_string_bounds, ignore_member_names, and ignore_enum_literal_names. If the DataReader does not allow type coercion, then its type must be equivalent to the type of the DataWriter.

Step 2. If either the DataWriter or the DataReader does not provide a TypeObject definition, then the registered type names are examined. The DataReader's and DataWriter's registered type names must match exactly, as was true in RTI Connext releases prior to 5.0.0.

If either Step 1 or Step 2 fails, the Topics associated with the DataReader and DataWriter are considered to be inconsistent and the **DDS_InconsistentTopicStatus** (p. 1165) is updated.

The default enforcement kind is **DDS_AUTO_TYPE_COERCION** (p. 651). This default kind translates to **DDS_ALLOW_TYPE_COERCION** (p. 651) except in the following cases:

- When a **Zero Copy** (p. 73) DataReader is used, the kind is translated to **DDS_DISALLOW_TYPE_COERCION** (p. 651).
- When the middleware is introspecting the built-in topic data declaration of a remote DataReader in order to determine whether it can match with a local DataWriter, if it observes that no TypeConsistencyEnforcementQosPolicy value is provided (as would be the case when communicating with a Service implementation not in conformance with this specification), it assumes a kind of **DDS_DISALLOW_TYPE_COERCION** (p. 651).

For additional information on type consistency enforcement refer to the `Extensible Types Guide` and the `OMG Extensible and Dynamic Topic Types for DDS Specification`.

Entity:

DDSDataReader (p. 1556)

Properties:

RxO (p. 556) = N/A
Changeable (p. 556) = **UNTIL ENABLE** (p. 556)

9.217.2 Member Data Documentation

9.217.2.1 kind

DDS_TypeConsistencyKind DDS_TypeConsistencyEnforcementQosPolicy::kind

<<***spec***>> *(p. 209)* Type consistency kind.

[default] DDS_AUTO_TYPE_COERCION (p. 651)

Categories

Immutable (p. 333), **Matching** (p. 382), **Discovery Metadata** (p. 377), **Type System** (p. 331)

9.217.2.2 ignore_sequence_bounds

DDS_Boolean DDS_TypeConsistencyEnforcementQosPolicy::ignore_sequence_bounds

<<***spec***>> *(p. 209)* Controls whether sequence bounds are taken into consideration for type assignability.

If the option is set to **DDS_BOOLEAN_TRUE** (p. 513), then sequence bounds (maximum lengths) are not considered as part of the type assignability. This means that a T2 sequence type with maximum length L2 would be assignable to a T1 sequence type with maximum length L1, even if L2 is greater than L1. If the option is set to **DDS_BOOLEAN_FALSE** (p. 513), then sequence bounds are taken into consideration for type assignability, and in order for T1 to be assignable from T2, it is required that L1 >= L2.

[default] DDS_BOOLEAN_TRUE (p. 513)

Categories

Immutable (p. 333), **Matching** (p. 382), **Discovery Metadata** (p. 377), **Type System** (p. 331)

9.217.2.3 ignore_string_bounds

DDS_Boolean DDS_TypeConsistencyEnforcementQosPolicy::ignore_string_bounds

<<***spec***>> *(p. 209)* Controls whether string bounds are taken into consideration for type assignability.

If the option is set to **DDS_BOOLEAN_TRUE** (p. 513), then string bounds (maximum lengths) are not considered as part of the type assignability. This means that a T2 string type with maximum length L2 would be assignable to a T1 string type with maximum length L1, even if L2 is greater than L1. If the option is set to **DDS_BOOLEAN_FALSE** (p. 513), then string bounds are taken into consideration for type assignability, and in order for T1 to be assignable from T2, it is required that L1 >= L2.

[default] DDS_BOOLEAN_TRUE (p. 513)

Categories

Immutable (p. 333), **Matching** (p. 382), **Discovery Metadata** (p. 377), **Type System** (p. 331)

9.217.2.4 ignore_member_names

DDS_Boolean DDS_TypeConsistencyEnforcementQosPolicy::ignore_member_names

<<***spec***>> *(p. 209)* Controls whether member names are taken into consideration for type assignability.

If the option is set to **DDS_BOOLEAN_TRUE** (p. 513), then member names are not considered as part of the type assignability. If the option is set to **DDS_BOOLEAN_FALSE** (p. 513), then member names are taken into consideration for type assignability, and in order for members with the same ID to be assignable, the members must also have the same name.

[default] DDS_BOOLEAN_FALSE (p. 513)

Categories

Immutable (p. 333), **Matching** (p. 382), **Discovery Metadata** (p. 377), **Type System** (p. 331)

9.217.2.5 prevent_type_widening

DDS_Boolean DDS_TypeConsistencyEnforcementQosPolicy::prevent_type_widening

<<***spec***>> *(p. 209)* Controls whether type widening is allowed.

If the option is set to **DDS_BOOLEAN_FALSE** (p. 513), then type widening is permitted. If the option is set to **DDS_↩ BOOLEAN_TRUE** (p. 513), then a wider type may not be assignable from a narrower type.

[default] DDS_BOOLEAN_FALSE (p. 513)

Categories

Immutable (p. 333), **Matching** (p. 382), **Discovery Metadata** (p. 377), **Type System** (p. 331)

9.217.2.6 force_type_validation

DDS_Boolean DDS_TypeConsistencyEnforcementQosPolicy::force_type_validation

<<***spec***>> *(p. 209)* Controls whether type information must be available in order to complete matching between a **DDSDataWriter** (p. 1589) and a **DDSDataReader** (p. 1556).

If the option is set to **DDS_BOOLEAN_TRUE** (p. 513), then type information must be available in order to complete matching between a **DDSDataWriter** (p. 1589) and a **DDSDataReader** (p. 1556). If the option is set to **DDS_↩ BOOLEAN_FALSE** (p. 513), then matching can occur without complete type information as long as the type names match exactly. Note that if the types have the same name but are not assignable, DataReaders may fail to deserialize incoming data samples.

[default] DDS_BOOLEAN_FALSE (p. 513)

Categories

Immutable (p. 333), **Matching** (p. 382), **Discovery Metadata** (p. 377), **Type System** (p. 331)

9.217.2.7 ignore_enum_literal_names

```
DDS_Boolean DDS_TypeConsistencyEnforcementQosPolicy::ignore_enum_literal_names
```

<<***spec***>> *(*p. *209)* Controls whether enumeration constant names are taken into consideration for type assignability.

If the option is set to **DDS_BOOLEAN_TRUE** (p. 513), then enumeration constants may change their names, but not their values, and still maintain assignability. If the option is set to **DDS_BOOLEAN_FALSE** (p. 513), then in order for enumerations to be assignable, any constant that has the same value in both enumerations must also have the same name.

[default] DDS_BOOLEAN_FALSE (p. 513)

Categories

Immutable (p. 333), **Matching** (p. 382), **Discovery Metadata** (p. 377), **Type System** (p. 331)

9.218 DDS_TypeDeallocationParams_t Struct Reference

Configures whether to release or not pointer and optional members.

Public Member Functions

- struct **DDS_TypeDeallocationParams_t** & **set_delete_pointers** (**DDS_Boolean** do_delete)
- struct **DDS_TypeDeallocationParams_t** & **set_delete_optional_members** (**DDS_Boolean** do_delete)

Public Attributes

- **DDS_Boolean delete_pointers**

  Whether to delete pointer members or not.
- **DDS_Boolean delete_optional_members**

  Whether to delete optional members or not.

9.218.1 Detailed Description

Configures whether to release or not pointer and optional members.

By default both pointers optional members are released when they are not null.

This structure is also defined in xcdr.1.0/srcC/typeObject/TypeObjectInfrastructure.h. If changes are made, please ensure they are made in both locations.

Examples

HelloWorld.cxx, and **HelloWorldPlugin.cxx**.

9.218.2 Member Function Documentation

9.218.2.1 set_delete_pointers()

```
struct DDS_TypeDeallocationParams_t & DDS_TypeDeallocationParams_t::set_delete_pointers (
            DDS_Boolean do_delete ) [inline]
```

Whether to delete pointer members or not

9.218.2.2 set_delete_optional_members()

```
struct DDS_TypeDeallocationParams_t & DDS_TypeDeallocationParams_t::set_delete_optional_members (
            DDS_Boolean do_delete ) [inline]
```

Whether to delete optional members or not

9.218.3 Member Data Documentation

9.218.3.1 delete_pointers

```
DDS_Boolean DDS_TypeDeallocationParams_t::delete_pointers
```

Whether to delete pointer members or not.

Examples

HelloWorld.cxx, and **HelloWorldPlugin.cxx**.

9.218.3.2 delete_optional_members

```
DDS_Boolean DDS_TypeDeallocationParams_t::delete_optional_members
```

Whether to delete optional members or not.

Examples

HelloWorld.cxx, and **HelloWorldPlugin.cxx**.

9.219 DDS_TypeSupportQosPolicy Struct Reference

[DEPRECATED] (p. **384)** <<***extension***>> *(*p. *213)* Allows you to attach application-specific values to a **DDSData↩ Writer** (p. 1589) or **DDSDataReader** (p. 1556), which are passed to the serialization or deserialization routine of the associated data type and choose whether to set padding bytes to zero during serialization.

Public Attributes

- void ∗ **plugin_data**

  [DEPRECATED] (p. ***384)*** <<***extension***>> *(*p. *213) Value to pass into the type plugin's de-/serialization function*
- **DDS_CdrPaddingKind cdr_padding_kind**

  [DEPRECATED] (p. ***384)*** <<***extension***>> *(*p. *213) Determines whether or not the padding bytes will be set to zero during CDR serialization.*

9.219.1 Detailed Description

[DEPRECATED] (p. **384)** <<***extension***>> *(*p. *213)* Allows you to attach application-specific values to a **DDSData↩ Writer** (p. 1589) or **DDSDataReader** (p. 1556), which are passed to the serialization or deserialization routine of the associated data type and choose whether to set padding bytes to zero during serialization.

The purpose of this QoS is to allow a user application to pass data to a type plugin's support functions and choose whether or not to set the padding bytes to zero when serializing a sample using CDR encapsulation.

Entity:

DDSDomainParticipant (p. 1619), **DDSDataReader** (p. 1556), **DDSDataWriter** (p. 1589)

Properties:

RxO (p. 556) = NO

Changeable (p. 556) = **UNTIL ENABLE** (p. 556)

9.219.2 Usage

The **DDS_TypeSupportQosPolicy::plugin_data** (p. 1500) allows you to associate a pointer to an object with a **DDSDataWriter** (p. 1589) or **DDSDataReader** (p. 1556). This object pointer is passed to the serialization routine of the data type associated with the **DDSDataWriter** (p. 1589) or the deserialization routine of the data type associated with the **DDSDataReader** (p. 1556).

You can modify the rtiddsgen-generated code so that the de/serialization routines act differently depending on the information passed in via the object pointer. (The generated serialization and deserialization code does not use the pointer.)

This functionality can be used to change how data sent by a **DDSDataWriter** (p. 1589) or received by a **DDSDataReader** (p. 1556) is serialized or deserialized on a per DataWriter and DataReader basis.

It can also be used to dynamically change how serialization (or for a less common case, deserialization) occurs. For example, a data type could represent a table, including the names of the rows and columns. However, since the row/column names of an instance of the table (a Topic) don't change, they only need to be sent once. The information passed in through the TypeSupport QoS policy could be used to signal the serialization routine to send the row/column names the first time a **DDSDataWriter** (p. 1589) calls **FooDataWriter::write** (p. 1951), and then never again.

The **DDS_TypeSupportQosPolicy::cdr_padding_kind** (p. 1500) allows you to choose whether or not the padding bytes are set to zero during CDR serialization.

9.219.3 Member Data Documentation

9.219.3.1 plugin_data

```
void* DDS_TypeSupportQosPolicy::plugin_data
```

[DEPRECATED] (p. **384)** <<***extension***>> *(*p. *213)* Value to pass into the type plugin's de-/serialization function

[default] NULL

Categories

Immutable (p. 333), **Application-Defined Metadata** (p. 254)

9.219.3.2 cdr_padding_kind

```
DDS_CdrPaddingKind DDS_TypeSupportQosPolicy::cdr_padding_kind
```

[DEPRECATED] (p. **384)** <<***extension***>> *(*p. *213)* Determines whether or not the padding bytes will be set to zero during CDR serialization.

In a DomainParticipant, this value configures how the padding bytes are set when serializing data for the Built-In Topic DataWriters and DataReaders. A value of **DDS_AUTO_CDR_PADDING** (p. 652) defaults to **DDS_NOT_SET_CDR_↩ PADDING** (p. 652).

For DataWriters and DataReaders, this value configures how padding bytes are set when serializing data for that entity. A value of **DDS_AUTO_CDR_PADDING** (p. 652) means that the entity will inherit whatever value is set on the Domain↩ Participant.

[default] DDS_AUTO_CDR_PADDING (p. 652)

Categories

Immutable (p. 333), **Security** (p. 325)

9.220 DDS_UInt8Seq Struct Reference

Instantiates **FooSeq** (p. 1965) < **DDS_UInt8** (p. 514) >

9.220.1 Detailed Description

Instantiates **FooSeq** (p. 1965) < **DDS_UInt8** (p. 514) >

Instantiates:

<<*generic*>> *(p. 433)* **FooSeq** (p. 1965)

See also

DDS_UInt8 (p. 514)

FooSeq (p. 1965)

9.221 DDS_UnionMember Struct Reference

A description of a member of a union.

Public Attributes

- char * **name**

  The name of the union member.
- **DDS_Boolean is_pointer**

  Indicates whether the union member is a pointer or not.
- struct **DDS_LongSeq labels**

  The labels of the union member.
- const **DDS_TypeCode** * **type**

  The type of the union member.

9.221.1 Detailed Description

A description of a member of a union.

See also

DDS_UnionMemberSeq (p. 1502)

DDS_TypeCodeFactory::create_union_tc (p. 1483)

9.221.2 Member Data Documentation

9.221.2.1 name

```
char* DDS_UnionMember::name
```

The name of the union member.

Cannot be NULL.

9.221.2.2 is_pointer

```
DDS_Boolean DDS_UnionMember::is_pointer
```

Indicates whether the union member is a pointer or not.

9.221.2.3 labels

```
struct  DDS_LongSeq DDS_UnionMember::labels
```

The labels of the union member.

Each union member should contain at least one label. If the union discriminator type is not **DDS_Long** (p. 515) the label value should be evaluated to an integer value. For instance, 'a' would be evaluated to 97.

9.221.2.4 type

```
const  DDS_TypeCode* DDS_UnionMember::type
```

The type of the union member.

Cannot be NULL.

9.222 DDS_UnionMemberSeq Struct Reference

Defines a sequence of union members.

9.222.1 Detailed Description

Defines a sequence of union members.

See also

DDS_UnionMember (p. 1501)
FooSeq (p. 1965)
DDS_TypeCodeFactory::create_union_tc (p. 1483)

9.223 DDS_UnsignedLongLongSeq Struct Reference

Instantiates **FooSeq** (p. 1965) < **DDS_UnsignedLongLong** (p. 515) >

9.223.1 Detailed Description

Instantiates **FooSeq** (p. 1965) < **DDS_UnsignedLongLong** (p. 515) >

Instantiates:

<<***generic***>> *(*p. *433)* **FooSeq** (p. 1965)

See also

DDS_UnsignedLongLong (p. 515)

FooSeq (p. 1965)

9.224 DDS_UnsignedLongSeq Struct Reference

Instantiates **FooSeq** (p. 1965) < **DDS_UnsignedLong** (p. 515) >

9.224.1 Detailed Description

Instantiates **FooSeq** (p. 1965) < **DDS_UnsignedLong** (p. 515) >

Instantiates:

<<***generic***>> *(*p. *433)* **FooSeq** (p. 1965)

See also

DDS_UnsignedLong (p. 515)

FooSeq (p. 1965)

9.225 DDS_UnsignedShortSeq Struct Reference

Instantiates **FooSeq** (p. 1965) < **DDS_UnsignedShort** (p. 514) >

9.225.1 Detailed Description

Instantiates **FooSeq** (p. 1965) < **DDS_UnsignedShort** (p. 514) >

Instantiates:

<<***generic***>> *(*p. *433)* **FooSeq** (p. 1965)

See also

DDS_UnsignedShort (p. 514)

FooSeq (p. 1965)

9.226 DDS_UserDataQosPolicy Struct Reference

Attaches a buffer of opaque data that is distributed by means of **Built-in Topics** (p. 61) during discovery.

Public Attributes

- struct **DDS_OctetSeq value**
  <<***spec***>> *(*p. *209) a sequence of octets*

9.226.1 Detailed Description

Attaches a buffer of opaque data that is distributed by means of **Built-in Topics** (p. 61) during discovery.

Entity:

DDSDomainParticipant (p. 1619), **DDSDataReader** (p. 1556), **DDSDataWriter** (p. 1589)

Properties:

RxO (p. 556) = NO;
Changeable (p. 556) = **YES** (p. 556)

See also

DDSDomainParticipant::get_builtin_subscriber (p. 1661)

9.226.2 Usage

The purpose of this QoS is to allow the application to attach additional information to the created **DDSEntity** (p. 1731) objects, so that when a remote application discovers their existence, it can access that information and use it for its own purposes. This information is not used by RTI Connext.

One possible use of this QoS is to attach security credentials or some other information that can be used by the remote application to authenticate the source.

In combination with operations such as **DDSDomainParticipant::ignore_participant** (p. 1661), **DDSDomain↩ Participant::ignore_publication** (p. 1664), **DDSDomainParticipant::ignore_subscription** (p. 1664), and **DDSDomain↩ Participant::ignore_topic** (p. 1663), this QoS policy can assist an application to define and enforce its own security policies.

The use of this QoS is not limited to security; it offers a simple, yet flexible extensibility mechanism.

Important: RTI Connext stores the data placed in this policy in pre-allocated pools. It is therefore necessary to configure RTI Connext with the maximum size of the data that will be stored in policies of this type. This size is configured with **DDS_DomainParticipantResourceLimitsQosPolicy::participant_user_data_max_length** (p. 991), **DDS_DomainParticipantResourceLimitsQosPolicy::writer_user_data_max_length** (p. 992), and **DDS_Domain↩ ParticipantResourceLimitsQosPolicy::reader_user_data_max_length** (p. 993).

9.226.3 Member Data Documentation

9.226.3.1 value

`struct` **`DDS_OctetSeq`** `DDS_UserDataQosPolicy::value`

<<***spec***>> *(p. 209)* a sequence of octets

[default] empty (zero-length)

[range] Octet sequence of length [0,max_length]

Categories

Mutable (p. 369), **Application-Defined Metadata** (p. 254), **Discovery Metadata** (p. 377)

9.227 DDS_ValueMember Struct Reference

A description of a member of a value type.

Public Attributes

- char * **name**

  The name of the value member.
- const **DDS_TypeCode** * **type**

  The type of the value member.
- **DDS_Boolean is_pointer**

  Indicates whether the value member is a pointer or not.
- **DDS_Short bits**

  Number of bits of a bitfield member.
- **DDS_Boolean is_key**

  Indicates if the value member is a key member or not.
- **DDS_Visibility access**

  The type of access (public, private) for the value member.
- **DDS_Long id**

  The member ID.
- **DDS_Boolean is_optional**

  Indicates if the value member is optional or required.
- **DDS_Boolean is_must_understand**

  Indicates if the value member is annotated as "must understand".

9.227.1 Detailed Description

A description of a member of a value type.

See also

DDS_ValueMemberSeq (p. 1508)

DDS_TypeCodeFactory::create_value_tc (p. 1482)

9.227.2 Member Data Documentation

9.227.2.1 name

```
char* DDS_ValueMember::name
```

The name of the value member.

Cannot be NULL.

9.227.2.2 type

```
const  DDS_TypeCode* DDS_ValueMember::type
```

The type of the value member.

Cannot be NULL.

9.227.2.3 is_pointer

```
DDS_Boolean DDS_ValueMember::is_pointer
```

Indicates whether the value member is a pointer or not.

9.227.2.4 bits

```
DDS_Short DDS_ValueMember::bits
```

Number of bits of a bitfield member.

If the struct member is a bitfield, this field contains the number of bits of the bitfield. Otherwise, bits should contain **DDS_TYPECODE_NOT_BITFIELD** (p. 83).

9.227.2.5 is_key

```
DDS_Boolean DDS_ValueMember::is_key
```

Indicates if the value member is a key member or not.

9.227.2.6 access

```
DDS_Visibility DDS_ValueMember::access
```

The type of access (public, private) for the value member.

It can take the values: **DDS_PRIVATE_MEMBER** (p. 85) or **DDS_PUBLIC_MEMBER** (p. 85).

9.227.2.7 id

```
DDS_Long DDS_ValueMember::id
```

The member ID.

Use **DDS_TYPECODE_MEMBER_ID_INVALID** (p. 83) to have the member ID automatically assigned.

9.227.2.8 is_optional

```
DDS_Boolean DDS_ValueMember::is_optional
```

Indicates if the value member is optional or required.

9.227.2.9 is_must_understand

```
DDS_Boolean DDS_ValueMember::is_must_understand
```

Indicates if the value member is annotated as "must understand".

9.228 DDS_ValueMemberSeq Struct Reference

Defines a sequence of value members.

9.228.1 Detailed Description

Defines a sequence of value members.

See also

DDS_ValueMember (p. 1505)
FooSeq (p. 1965)
DDS_TypeCodeFactory::create_value_tc (p. 1482)

9.229 DDS_VendorId_t Struct Reference

<<***extension***>> *(p. 432)* Type used to represent the vendor of the service implementing the RTPS protocol.

Public Attributes

- **DDS_Octet vendorId** [**DDS_VENDOR_ID_LENGTH_MAX**]
  The vendor Id.

9.229.1 Detailed Description

<<***extension***>> *(p. 432)* Type used to represent the vendor of the service implementing the RTPS protocol.

9.229.2 Member Data Documentation

9.229.2.1 vendorId

```
DDS_Octet DDS_VendorId_t::vendorId[ DDS_VENDOR_ID_LENGTH_MAX]
```

The vendor Id.

9.230 DDS_VirtualWriterInfo Struct Reference

The structure that contains the information of a virtual writer.

9.230.1 Detailed Description

The structure that contains the information of a virtual writer.

9.231 DDS_WaitSetProperty_t Struct Reference

<<***extension***>> *(*p. *432)* Specifies the **DDSWaitSet** (p. 1897) behavior for multiple trigger events.

Public Attributes

- **DDS_Long max_event_count**
  *Maximum number of trigger events to cause a **DDSWaitSet** (*p. *1897) to awaken.*
- struct **DDS_Duration_t max_event_delay**
  *Maximum delay from occurrence of first trigger event to cause a **DDSWaitSet** (*p. *1897) to awaken.*

9.231.1 Detailed Description

<<***extension***>> *(*p. *432)* Specifies the **DDSWaitSet** (p. 1897) behavior for multiple trigger events.

In simple use, a **DDSWaitSet** (p. 1897) returns when a single trigger event occurs on one of its attached **DDSCondition** (p. 1544) (s), or when the `timeout` maximum wait duration specified in the **DDSWaitSet::wait** (p. 1901) call expires.

The **DDS_WaitSetProperty_t** (p. 1509) allows configuration of the waiting behavior of a **DDSWaitSet** (p. 1897). If no conditions are true at the time of the call to wait, then the `max_event_count` parameter may be used to configure the WaitSet to wait for `max_event_count` trigger events to occur before returning, or to wait for up to `max_event_delay` time from the occurrence of the first trigger event before returning.

The `timeout` maximum wait duration specified in the **DDSWaitSet::wait** (p. 1901) call continues to apply.

Entity:

DDSWaitSet (p. 1897)

Properties:

RxO (p. 556) = N/A

Changeable (p. 556) = **YES** (p. 556)

9.231.2 Member Data Documentation

9.231.2.1 max_event_count

DDS_Long DDS_WaitSetProperty_t::max_event_count

Maximum number of trigger events to cause a **DDSWaitSet** (p. 1897) to awaken.

The **DDSWaitSet** (p. 1897) will wait until up to max_event_count trigger events have occurred before returning. The **DDSWaitSet** (p. 1897) may return earlier if either the timeout duration has expired, or max_event_delay has elapsed since the occurrence of the first trigger event. max_event_count may be used to "collect" multiple trigger events for processing at the same time.

[default] 1

[range] >= 1

;

9.231.2.2 max_event_delay

struct **DDS_Duration_t** DDS_WaitSetProperty_t::max_event_delay

Maximum delay from occurrence of first trigger event to cause a **DDSWaitSet** (p. 1897) to awaken.

The **DDSWaitSet** (p. 1897) will return no later than max_event_delay after the first trigger event. max_event_↩ delay may be used to establish a maximum latency for events reported by the **DDSWaitSet** (p. 1897).

Note that **DDS_RETCODE_TIMEOUT** (p. 533) is *not* returned if max_event_delay is exceeded. **DDS_RETCODE**↩ **_TIMEOUT** (p. 533) is returned only if the timeout duration expires before any trigger events occur.

[default] DDS_DURATION_INFINITE (p. 522) ;

9.232 DDS_WcharSeq Struct Reference

Instantiates **FooSeq** (p. 1965) < **DDS_Wchar** (p. 514) >

9.232.1 Detailed Description

Instantiates **FooSeq** (p. 1965) < **DDS_Wchar** (p. 514) >

Instantiates:

<<***generic***>> *(*p. *433)* **FooSeq** (p. 1965)

See also

DDS_Wchar (p. 514)

FooSeq (p. 1965)

9.233 DDS_WireProtocolQosPolicy Struct Reference

<<***extension***>> *(*p. *213)* Specifies the wire-protocol-related attributes for the **DDSDomainParticipant** (p. 1619).

Public Attributes

- **DDS_Long participant_id**

  <<***basic***>> *(*p. *201)* <<***extension***>> *(*p. *213) A value used to distinguish among different participants belonging to the same domain on the same host.*
- **DDS_UnsignedLong rtps_host_id**

  <<***extension***>> *(*p. *213) The RTPS Host ID of the domain participant.*
- **DDS_UnsignedLong rtps_app_id**

  <<***extension***>> *(*p. *213) The RTPS App ID of the domain participant.*
- **DDS_UnsignedLong rtps_instance_id**

  <<***extension***>> *(*p. *213) The RTPS Instance ID of the* ***DDSDomainParticipant*** *(*p. *1619).*
- struct **DDS_RtpsWellKnownPorts_t rtps_well_known_ports**

  <<***extension***>> *(*p. *213) Configures the RTPS well-known port mappings.*
- **DDS_RtpsReservedPortKindMask rtps_reserved_port_mask**

  <<***extension***>> *(*p. *213) Specifies which well-known ports to reserve when enabling the participant.*
- **DDS_WireProtocolQosPolicyAutoKind rtps_auto_id_kind**

  <<***extension***>> *(*p. *213) Kind of auto mechanism used to calculate the GUID prefix.*
- **DDS_Boolean compute_crc**

  <<***extension***>> *(*p. *213) Adds RTPS CRC submessage to every message when this field is set to* ***DDS_BOOLEAN_TRUE*** *(*p. *513).*
- **DDS_Boolean check_crc**

  <<***extension***>> *(*p. *213) Checks if the received RTPS message is valid by comparing the computed CRC with the received RTPS CRC submessage when this field is set to* ***DDS_BOOLEAN_TRUE*** *(*p. *513).*

9.233.1 Detailed Description

<<***extension***>> *(p. 213)* Specifies the wire-protocol-related attributes for the **DDSDomainParticipant** (p. 1619).

Entity:

DDSDomainParticipant (p. 1619)

Properties:

RxO (p. 556) = N/A
Changeable (p. 556) = **NO** (p. 556)

9.233.2 Usage

This QoS policy configures some participant-wide properties of the DDS Real-Time Publish Subscribe (RTPS) on-the-wire protocol. (**DDS_DataWriterProtocolQosPolicy** (p. 892) and **DDS_DataReaderProtocolQosPolicy** (p. 843) configure RTPS and reliability properties on a per **DDSDataWriter** (p. 1589) or **DDSDataReader** (p. 1556) basis.)

NOTE: The default QoS policies returned by RTI Connext contain the correctly initialized wire protocol attributes. The defaults are not normally expected to be modified, but are available to the advanced user customizing the implementation behavior.

The default values should not be modified without an understanding of the underlying Real-Time Publish Subscribe (RTPS) wire protocol.

In order for the discovery process to work correctly, each **DDSDomainParticipant** (p. 1619) must have a unique identifier. This QoS policy specifies how that identifier should be generated.

RTPS defines a 96-bit prefix to this identifier; each **DDSDomainParticipant** (p. 1619) must have a unique value for this prefix relative to all other participants in its domain.

If an application dies unexpectedly and is restarted, the IDs used by the new instance of DomainParticipants should be different than the ones used by the previous instances. A change in these values allows other DomainParticipants to know that they are communicating with a new instance of an application, and not the previous instance.

For legacy reasons, RTI Connext divides the 96-bit prefix into three integers:

- The first integer is called host ID. The original purpose of this integer was to contain the identity of the machine on which the DomainParticipant is executing.
- The second integer is called an application ID. The original purpose of this integer was to contain a value that identifies the process or task in which the DomainParticipant is contained.
- The third integer is called instance ID. The original purpose was to contain a value that uniquely identifies a DomainParticipant within a task or process.

The **DDS_WireProtocolQosPolicy::rtps_auto_id_kind** (p. 1518) field can be used to configure the algorithm that RTI Connext uses to populate the 96-bit prefix. Then you can optionally overwrite specific parts of the 96-bit prefix by explicitly configuring the **DDS_WireProtocolQosPolicy::rtps_host_id** (p. 1515) (first integer), **DDS_WireProtocol↩ QosPolicy::rtps_app_id** (p. 1516) (second integer), and **DDS_WireProtocolQosPolicy::rtps_instance_id** (p. 1516) (third integer).

The **DDS_WireProtocolQosPolicy::rtps_auto_id_kind** (p. 1518) field supports three different prefix generation algorithms:

1. In the default and most common scenario, **DDS_WireProtocolQosPolicy::rtps_auto_id_kind** (p. 1518) is set to **DDS_RTPS_AUTO_ID_FROM_UUID** (p. 658). As the name suggests, this mechanism uses a unique, randomly generated UUID to fill the **rtps_host_id**, **rtps_app_id**, or **rtps_instance_id** fields. The first two bytes of the **rtps_host_id** are replaced with the RTI vendor ID (0x0101).

2. (Legacy) When rtps_auto_id_kind is set to **DDS_RTPS_AUTO_ID_FROM_IP** (p. 658), the 96-bit prefix is generated as follows:

   - **rtps_host_id:** The 32-bit value of the IPv4 of the first up and running interface of the host machine is assigned. If the host does not have an IPv4 address, the host-id will be automatically set to 0x7F000001.
   - **rtps_app_id:** The process (or task) ID is assigned.
   - **rtps_instance_id:** A counter is assigned that is incremented per new participant within a process.

   DDS_RTPS_AUTO_ID_FROM_IP is not a good algorithm to guarantee prefix uniqueness, because the process ID can be recycled by the OSs.

3. (Legacy) When rtps_auto_id_kind is set to **DDS_RTPS_AUTO_ID_FROM_MAC** (p. 658), the 96-bit prefix is generated as follows:

   - **rtps_host_id:** The first 32 bits of the MAC address of the first up and running interface of the host machine are assigned.
   - **rtps_app_id:** The last 32 bits of the MAC address of the first up and running interface of the host machine are assigned.
   - **rtps_instance_id:** This field is split into two different parts. The process (or task) ID is assigned to the first 24 bits. A counter is assigned to the last 8 bits. This counter is incremented per new participant.

   DDS_RTPS_AUTO_ID_FROM_IP is not a good algorithm to guarantee prefix uniqueness, because the process ID can be recycled by the OSs.

Some examples are provided to clarify the behavior of this QoSPolicy in case you want to change the default behavior with **DDS_RTPS_AUTO_ID_FROM_MAC** (p. 658).

First, get the participant QoS from the DomainParticipantFactory:

```
DDSDomainParticipantFactory *factory = NULL;

factory = DDSTheParticipantFactory->get_instance();

factory->get_default_participant_qos(participant_qos);
```

Second, change the **DDS_WireProtocolQosPolicy** (p. 1511) using one of the options shown below.

Third, create the **DDSDomainParticipant** (p. 1619) as usual, using the modified QoS structure instead of the default one.

Option 1: Use **DDS_RTPS_AUTO_ID_FROM_MAC** (p. 658) to explicitly set just the application/task identifier portion of the **rtps_instance_id** field.

```
participant_qos.wire_protocol.rtps_auto_id_kind = DDS_RTPS_AUTO_ID_FROM_MAC;

participant_qos.wire_protocol.rtps_host_id      = DDS_RTPS_AUTO_ID;

participant_qos.wire_protocol.rtps_app_id       = DDS_RTPS_AUTO_ID;

participant_qos.wire_protocol.rtps_instance_id  = (/* App ID */ (12 << 8) |

                                                  /* Instance ID*/ (DDS_RTPS_AUTO_ID));
```

Option 2: Handle only the per participant counter and let RTI Connext handle the application/task identifier:

```
participant_qos.wire_protocol.rtps_auto_id_kind = DDS_RTPS_AUTO_ID_FROM_MAC;

participant_qos.wire_protocol.rtps_host_id      = DDS_RTPS_AUTO_ID;

participant_qos.wire_protocol.rtps_app_id       = DDS_RTPS_AUTO_ID;

participant_qos.wire_protocol.rtps_instance_id  =  (/* App ID */ (DDS_RTPS_AUTO_ID) |

                                                  /* Instance ID*/ (12));
```

Option 3: Handle the entire **rtps_instance_id** field yourself:

```
participant_qos.wire_protocol.rtps_auto_id_kind = DDS_RTPS_AUTO_ID_FROM_MAC;

participant_qos.wire_protocol.rtps_host_id      = DDS_RTPS_AUTO_ID;

participant_qos.wire_protocol.rtps_app_id       = DDS_RTPS_AUTO_ID;

participant_qos.wire_protocol.rtps_instance_id  = ( /* App ID */ (12 << 8)) |

                                                   /* Instance ID */ (9) )
```

NOTE: If you are using **DDS_RTPS_AUTO_ID_FROM_MAC** (p. 658) as **rtps_auto_id_kind** and you decide to manually handle the **rtps_instance_id** field, you must ensure that both parts are non-zero (otherwise RTI Connext will take responsibility for them). RTI recommends that you always specify the two parts separately in order to avoid errors.

Option 4: Let RTI Connext handle the entire **rtps_instance_id** field:

```
participant_qos.wire_protocol.rtps_auto_id_kind = DDS_RTPS_AUTO_ID_FROM_MAC;

participant_qos.wire_protocol.rtps_host_id      = DDS_RTPS_AUTO_ID;

participant_qos.wire_protocol.rtps_app_id       = DDS_RTPS_AUTO_ID;

participant_qos.wire_protocol.rtps_instance_id  = DDS_RTPS_AUTO_ID;
```

NOTE: If you are using **DDS_RTPS_AUTO_ID_FROM_MAC** (p. 658) as **rtps_auto_id_kind** and you decide to manually handle the **rtps_instance_id** field, you must ensure that both parts are non-zero (otherwise RTI Connext will take responsibility for them). RTI recommends that you always specify the two parts separately in order to clearly show the difference.

9.233.3 Member Data Documentation

9.233.3.1 participant_id

DDS_Long DDS_WireProtocolQosPolicy::participant_id

<<***basic***>> *(p. 201)* <<***extension***>> *(p. 213)* A value used to distinguish among different participants belonging to the same domain on the same host.

Determines the unicast port on which meta-traffic is received. Also defines the *default* unicast port for receiving user-traffic for DataReaders and DataWriters (can be overridden by the **DDS_DataReaderQos::unicast** (p. 864) or **DDS_DataWriterQos::unicast** (p. 915)).

For more information on port mapping, please refer to **DDS_RtpsWellKnownPorts_t** (p. 1338).

Each **DDSDomainParticipant** (p. 1619) in the same domain and running on the same host, must have a unique `participant_id`. The participants may be in the same address space or in distinct address spaces.

A negative number (-1) means that RTI Connext will *automatically* resolve the participant ID as follows.

- RTI Connext will pick the *smallest* participant ID based on the unicast ports available on the transports enabled for discovery.
- RTI Connext will attempt to resolve an automatic port index either when a DomainPartcipant is enabled, or when a DataReader or DataWriter is created. Therefore, all the transports enabled for discovery must have been registered by this time. Otherwise, the discovery transports registered after resolving the automatic port index may produce port conflicts when the DomainPartcipant is enabled.

[default] -1 [automatic], i.e. RTI Connext will automatically pick the `participant_id`, as described above.

[range] [>= 0], or -1, and does not violate guidelines stated in **DDS_RtpsWellKnownPorts_t** (p. 1338).

See also

DDSEntity::enable() (p. 1734)

NDDSTransportSupport::register_transport() (p. 2104)

Categories

Immutable (p. 333), **Discovery** (p. 271)

9.233.3.2 rtps_host_id

```
DDS_UnsignedLong DDS_WireProtocolQosPolicy::rtps_host_id
```

<<***extension***>> *(*p. *213)* The RTPS Host ID of the domain participant.

A specific host ID that is unique in the domain.

[default] DDS_RTPS_AUTO_ID (p. **??**). The default value is interpreted as follows:
If **DDS_WireProtocolQosPolicy::rtps_auto_id_kind** (p. 1518) is equal to **DDS_RTPS_AUTO_ID_FROM_IP** (p. 658), the value will be interpreted as the IPv4 address of the *first* up and running interface of the host machine.
If **DDS_WireProtocolQosPolicy::rtps_auto_id_kind** (p. 1518) is equal to **DDS_RTPS_AUTO_ID_FROM_MAC** (p. 658), the value will be interpreted as the first 32 bits of the MAC address assigned to the *first* up and running interface of the host machine.
If **DDS_WireProtocolQosPolicy::rtps_auto_id_kind** (p. 1518) is equal to **DDS_RTPS_AUTO_ID_FROM_UUID** (p. 658), the value will be the first 32 bits of the GUID prefix assigned by the UUID algorithm.

[range] [0,0xffffffff]

Categories

Immutable (p. 333), **Discovery** (p. 271)

9.233.3.3 rtps_app_id

```
DDS_UnsignedLong DDS_WireProtocolQosPolicy::rtps_app_id
```

<<***extension***>> *(*p. *213)* The RTPS App ID of the domain participant.

A participant specific ID that, together with the rtps_instance_id, is unique within the scope of the rtps_host_id.

If a participant dies and is restarted, it is recommended that it be given an app ID that is distinct from the previous one so that other participants in the domain can distinguish between them.

[default] DDS_RTPS_AUTO_ID (p. **??**). The default value is interpreted as follows:
If **DDS_WireProtocolQosPolicy::rtps_auto_id_kind** (p. 1518) is equal to ***DDS_RTPS_AUTO_ID_FROM_IP*** *(*p. *658)* the value will be the process (or task) ID.
If **DDS_WireProtocolQosPolicy::rtps_auto_id_kind** (p. 1518) is equal to ***DDS_RTPS_AUTO_ID_FROM_MAC*** *(*p. *658)* the value will be the last 32 bits of the MAC address assigned to the *first* up and running interface of the host machine.
If **DDS_WireProtocolQosPolicy::rtps_auto_id_kind** (p. 1518) is equal to ***DDS_RTPS_AUTO_ID_FROM_UUID*** *(*p. *658)*, the value will be the middle 32 bits of the GUID prefix assigned by the UUID algorithm.

[range] [0,0xffffffff]

Categories

Immutable (p. 333), **Discovery** (p. 271)

9.233.3.4 rtps_instance_id

DDS_UnsignedLong DDS_WireProtocolQosPolicy::rtps_instance_id

<<***extension***>> *(*p. *213)* The RTPS Instance ID of the **DDSDomainParticipant** (p. 1619).

This is an instance-specific ID of a participant that, together with the rtps_app_id, is unique within the scope of the rtps_host_id.

If a participant dies and is restarted, it is recommended that it be given an instance ID that is distinct from the previous one so that other participants in the domain can distinguish between them.

[default] DDS_RTPS_AUTO_ID (p. **??**). The default value is interpreted as follows:
If **DDS_WireProtocolQosPolicy::rtps_auto_id_kind** (p. 1518) is equal to ***DDS_RTPS_AUTO_ID_FROM_IP*** *(*p. *658)*, a counter is assigned that is incremented per new participant. For VxWorks-653, the first 8 bits are assigned to the partition id for the application.
If **DDS_WireProtocolQosPolicy::rtps_auto_id_kind** (p. 1518) is equal to ***DDS_RTPS_AUTO_ID_FROM_MAC*** *(*p. *658)*, the first 24 bits are assigned to the application/task identifier and the last 8 bits are assigned to a counter that is incremented per new paricipant.
If **DDS_WireProtocolQosPolicy::rtps_auto_id_kind** (p. 1518) is equal to ***DDS_RTPS_AUTO_ID_FROM_UUID*** *(*p. *658)*, the value will be the last 32 bits of the GUID prefix assigned by the UUID algorithm.

[range] [0,0xffffffff] **NOTE:** If you use ***DDS_RTPS_AUTO_ID_FROM_MAC*** *(*p. *658)* as **rtps_auto_id_kind** and you decide to manually handle the **rtps_instance_id** field, you must ensure that both the two parts are non-zero, otherwise the middleware will take responsibility for them. We recommend that you always specify the two parts separately in order to avoid errors. (`examples`)

Categories

Immutable (p. 333), **Discovery** (p. 271)

9.233.3.5 rtps_well_known_ports

struct **DDS_RtpsWellKnownPorts_t** DDS_WireProtocolQosPolicy::rtps_well_known_ports

<<***extension***>> *(*p. *213)* Configures the RTPS well-known port mappings.

Determines the well-known multicast and unicast port mappings for discovery (meta) traffic and user traffic.

[default] DDS_INTEROPERABLE_RTPS_WELL_KNOWN_PORTS (p. 659)

Categories

Immutable (p. 333), **Discovery** (p. 271)

9.233.3.6 rtps_reserved_port_mask

DDS_RtpsReservedPortKindMask DDS_WireProtocolQosPolicy::rtps_reserved_port_mask

<<***extension***>> *(p. 213)* Specifies which well-known ports to reserve when enabling the participant.

Specifies which of the well-known multicast and unicast ports will be reserved when the DomainParticipant is enabled. Failure to allocate a port that is computed based on the **DDS_RtpsWellKnownPorts_t** (p. 1338) will be detected at this time, and the enable operation will fail.

[default] DDS_RTPS_RESERVED_PORT_MASK_DEFAULT (p. 655)

Categories

Immutable (p. 333), **Discovery** (p. 271)

9.233.3.7 rtps_auto_id_kind

DDS_WireProtocolQosPolicyAutoKind DDS_WireProtocolQosPolicy::rtps_auto_id_kind

<<***extension***>> *(p. 213)* Kind of auto mechanism used to calculate the GUID prefix.

[default] DDS_RTPS_AUTO_ID_FROM_UUID (p. 658)

Categories

Immutable (p. 333), **Discovery** (p. 271)

9.233.3.8 compute_crc

DDS_Boolean DDS_WireProtocolQosPolicy::compute_crc

<<***extension***>> *(p. 213)* Adds RTPS CRC submessage to every message when this field is set to **DDS_BOOLEAN↩_TRUE** (p. 513).

The computed CRC covers the entire RTPS message excluding the RTPS header.

[default] DDS_BOOLEAN_FALSE (p. 513)

Categories

Immutable (p. 333), **Security** (p. 325)

9.233.3.9 check_crc

```
DDS_Boolean DDS_WireProtocolQosPolicy::check_crc
```

<<***extension***>> *(p. 213)* Checks if the received RTPS message is valid by comparing the computed CRC with the received RTPS CRC submessage when this field is set to **DDS_BOOLEAN_TRUE** (p. 513).

DDS_WireProtocolQosPolicy::compute_crc (p. 1518) must be enabled at the publishing application for validating the message at the subscribing application.

[default] DDS_BOOLEAN_FALSE (p. 513)

Categories

Immutable (p. 333), **Security** (p. 325)

9.234 DDS_WriteParams_t Struct Reference

<<***extension***>> *(p. 432)* Input parameters for writing with **FooDataWriter::write_w_params** (p. 1956), **FooData↩ Writer::dispose_w_params** (p. 1959), **FooDataWriter::register_instance_w_params** (p. 1948), **FooDataWriter↩ ::unregister_instance_w_params** (p. 1951)

Public Attributes

- **DDS_Boolean replace_auto**

  Allows retrieving the actual value of those fields that were automatic.
- struct **DDS_SampleIdentity_t identity**

  Identity of the sample.
- struct **DDS_SampleIdentity_t related_sample_identity**

  The identity of another sample related to this one.
- struct **DDS_Time_t source_timestamp**

  Source timestamp upon write.
- struct **DDS_Cookie_t cookie**

  Octet sequence identifying written data sample.
- **DDS_InstanceHandle_t handle**

  Instance handle.
- **DDS_Long priority**

  Publication priority.
- **DDS_SampleFlag flag**

  Flags associated with the sample.
- struct **DDS_GUID_t source_guid**

  Identifies the application logical data source associated with the sample being written.
- struct **DDS_GUID_t related_source_guid**

  Identifies the application logical data source that is related to the sample being written.
- struct **DDS_GUID_t related_reader_guid**

  Identifies a DataReader that is logically related to the sample that is being written.

9.234.1 Detailed Description

<<***extension***>> *(p. 432)* Input parameters for writing with **FooDataWriter::write_w_params** (p. 1956), **FooData↩ Writer::dispose_w_params** (p. 1959), **FooDataWriter::register_instance_w_params** (p. 1948), **FooDataWriter↩ ::unregister_instance_w_params** (p. 1951)

9.234.2 Member Data Documentation

9.234.2.1 replace_auto

```
DDS_Boolean DDS_WriteParams_t::replace_auto
```

Allows retrieving the actual value of those fields that were automatic.

When this field is set, the fields that were configured with an automatic value (for example, **DDS_AUTO_SAMPLE_↩ IDENTITY** (p. 676)) receive their actual value after **FooDataWriter::write_w_params** (p. 1956) is called.

To reset those fields to their automatic value after calling **FooDataWriter::write_w_params** (p. 1956), use **DDS_↩ WriteParams_reset** (p. 676)

9.234.2.2 identity

```
struct DDS_SampleIdentity_t DDS_WriteParams_t::identity
```

Identity of the sample.

Identifies the sample being written. The identity consists of a pair (Virtual Writer GUID, Virtual Sequence Number).

Use the default value to let RTI Connext determine the sample identity as follows:

- The Virtual Writer GUID is the virtual GUID associated with the writer writing the sample. This virtual GUID is configured using **DDS_DataWriterProtocolQosPolicy::virtual_guid** (p. 893).
- The sequence number is increased by one with respect to the previous value.

The virtual sequence numbers for a virtual writer must be strictly monotonically increasing. If the user tries to write a sample with a sequence number smaller or equal to the last sequence number, the write operation will fail.

A DataReader can access the identity of a received sample by using the fields **DDS_SampleInfo::original↩ _publication_virtual_guid** (p. 1352) and **DDS_SampleInfo::original_publication_virtual_sequence_number** (p. 1352) in the **DDS_SampleInfo** (p. 1344).

[default] DDS_AUTO_SAMPLE_IDENTITY (p. 676).

Referenced by **connext::WriteSample< T >::identity()**, and **connext::WriteSampleRef< T >::identity()**.

9.234.2.3 related_sample_identity

```
struct DDS_SampleIdentity_t DDS_WriteParams_t::related_sample_identity
```

The identity of another sample related to this one.

Identifies another sample that is logically related to the one that is written.

When this field is set, the related sample identity is propagated and subscribing applications can retrieve it from the **DDS_SampleInfo** (p. 1344) (see **DDS_SampleInfo_get_related_sample_identity** (p. 154)).

The default value is **DDS_UNKNOWN_SAMPLE_IDENTITY** (p. 676), and is not propagated.

A DataReader can access the related identity of a received sample by using the fields **DDS_SampleInfo::related_original_publication_virtual_guid** (p. 1352) and **DDS_SampleInfo::related_original_publication_virtual_sequence_number** (p. 1353) in the **DDS_SampleInfo** (p. 1344).

[default] DDS_UNKNOWN_SAMPLE_IDENTITY (p. 676)

9.234.2.4 source_timestamp

```
struct DDS_Time_t DDS_WriteParams_t::source_timestamp
```

Source timestamp upon write.

Specifies the source timestamp that will be available to the **DDSDataReader** (p. 1556) objects by means of the `source_timestamp` attribute within the **DDS_SampleInfo** (p. 1344).

[default] DDS_TIME_INVALID (p. 522).

9.234.2.5 cookie

```
struct DDS_Cookie_t DDS_WriteParams_t::cookie
```

Octet sequence identifying written data sample.

Used in the callback **DDSDataWriterListener::on_sample_removed** (p. 1616) to associate a removed sample with a written sample.

[default] Empty sequence (zero-length).

9.234.2.6 handle

```
DDS_InstanceHandle_t DDS_WriteParams_t::handle
```

Instance handle.

Either the handle returned by a previous call to **FooDataWriter::register_instance** (p. 1946), or else the special value **DDS_HANDLE_NIL** (p. 78).

[default] DDS_HANDLE_NIL (p. 78)

9.234.2.7 priority

DDS_Long DDS_WriteParams_t::priority

Publication priority.

A positive integer value designating the relative priority of the sample, used to determine the transmission order of pending writes.

Use of publication priorities requires an asynchronous publisher (**DDS_ASYNCHRONOUS_PUBLISH_MODE_QOS** (p. 629)) with **DDS_FlowControllerProperty_t::scheduling_policy** (p. 1157) set to **DDS_HPF_FLOW_CONTROLLER_SCHED_POLICY** (p. 125).

Larger numbers have higher priority.

For multi-channel DataWriters, the publication priority of a sample may be used as a filter criteria for determining channel membership.

If the publication priority of the parent DataWriter, or for multi-channel DataWriters, if the publication priority of the parent channel, is set to **DDS_PUBLICATION_PRIORITY_AUTOMATIC** (p. 628), then the DataWriter or channel will be assigned the priority of the largest publication priority of all samples in the DataWriter or channel.

If the publication priority of the parent DataWriter, and for multi-channel DataWriters, if the publication priority of the parent channel, are set to **DDS_PUBLICATION_PRIORITY_UNDEFINED** (p. 628), then the DataWriter or channel will be assigned the lowest priority, regardless of the value of the publication priorities of samples written to the DataWriter or channel.

The publication priority of each sample can be set in the **DDS_WriteParams_t** (p. 1519) of **FooDataWriter::write_w_params** (p. 1956).

For dispose and unregister samples, use the **DDS_WriteParams_t** (p. 1519) of **FooDataWriter::dispose_w_params** (p. 1959) and **FooDataWriter::unregister_instance_w_params** (p. 1951).

[default] 0 (lowest priority)

See also

DDS_ChannelSettings_t::priority (p. 823)

9.234.2.8 flag

DDS_SampleFlag DDS_WriteParams_t::flag

Flags associated with the sample.

The flags are represented as a 32-bit integer, of which only the 16 least-significant bits are used.

RTI reserves least-significant bits [0-7] for middleware-specific usage.

The application can use least-significant bits [8-15].

The first bit (**DDS_REDELIVERED_SAMPLE** (p. 673)) is reserved for marking samples as redelivered when using RTI Queuing Service.

The second bit (**DDS_INTERMEDIATE_REPLY_SEQUENCE_SAMPLE** (p. 673)) is used to indicate that a response sample is not the last response sample for a given request. This bit is usually set by a Replier sending multiple responses for a request.

An application can inspect the flags associated with a received sample by checking the field **DDS_SampleInfo::flag** (p. 1353).

[default] 0 (no flags are set)

9.234.2.9 source_guid

```
struct DDS_GUID_t DDS_WriteParams_t::source_guid
```

Identifies the application logical data source associated with the sample being written.

When this field is set, the source_guid is propagated and subscribing applications can retrieve it from the **DDS_SampleInfo** (p. 1344) (see **DDS_SampleInfo::source_guid** (p. 1353)).

The default value is **DDS_GUID_AUTO** (p. 527), and is not propagated.

This field is used by RTI Connext in the following cases:

- In RTI Queuing Service, this field and **DDS_WriteParams_t::related_source_guid** (p. 1523) play a crucial role in ensuring that replies are correctly routed back to the requester in request/reply scenarios.
- In RTI Routing Service, this field is used by the TopicQuery proxy feature to identify the original writer(s) that published the response samples for the TopicQuery. When dealing with snapshot TopicQueries, there will be one final sample (sample without the flag **DDS_INTERMEDIATE_TOPIC_QUERY_SAMPLE** (p. 674)) per original writer that published one or more response samples.

You can use this field for your own purposes unless you are using Queuing Service or Routing Service as described above.

[default] DDS_GUID_AUTO (p. 527) (the source_guid is automatically set to the **DDSDataWriter** (p. 1589) virtual GUID).

See also

DDS_WriteParams_t::related_source_guid (p. 1523)

9.234.2.10 related_source_guid

```
struct DDS_GUID_t DDS_WriteParams_t::related_source_guid
```

Identifies the application logical data source that is related to the sample being written.

When this field is set, the related_source_guid is propagated and subscribing applications can retrieve it from the **DDS_SampleInfo** (p. 1344) (see **DDS_SampleInfo::related_source_guid** (p. 1353)).

This field and **DDS_WriteParams_t::source_guid** (p. 1522) are used by RTI Queuing Service, to ensure that replies are correctly routed back to the requester in request/reply scenarios.

The default value is **DDS_GUID_UNKNOWN** (p. 527), and is not propagated.

[default] DDS_GUID_UNKNOWN (p. 527)

See also

DDS_WriteParams_t::source_guid (p. 1522)

9.234.2.11 related_reader_guid

```
struct DDS_GUID_t DDS_WriteParams_t::related_reader_guid
```

Identifies a DataReader that is logically related to the sample that is being written.

When this field is set, the related_reader_guid is propagated and subscribing applications can retrieve it from the **DDS_SampleInfo** (p. 1344) (see **DDS_SampleInfo::related_subscription_guid** (p. 1354)).

The default value is **DDS_GUID_UNKNOWN** (p. 527), and is not propagated.

The main use case for this field is point-to-point sample distribution using CFT. DataReaders install a CFT on the related_reader_guid using a unique GUID. For example, the filter for DataReader 'n' can be:
`@related_reader_guid.value = &hex(00000000000000000000000000000001)`

Then, a DataWriter that wants to send the sample to DataReader 'n' will use the **FooDataWriter::write_w_params** (p. 1956) method and set related_reader_guid to the value used by DataReader 'n' in its filter expression.

This field is currently used by RTI Queuing Service to distribute a sample to only one of the Consumer's DataReaders attached to a SharedReaderQueue.

[default] DDS_GUID_UNKNOWN (p. 527)

9.235 DDS_WriterDataLifecycleQosPolicy Struct Reference

*<<**spec**>> (p. 209)* Controls how a **DDSDataWriter** (p. 1589) handles the lifecycle of the instances (keys) that it is registered to manage.

Public Attributes

- **DDS_Boolean autodispose_unregistered_instances**

  *<<**spec**>> (p. 209) Boolean flag that controls whether or not a writer will also dispose of an instance when the **DDSDataWriter** (p. 1589) unregisters from the instance by means of the unregister operations.*
- struct **DDS_Duration_t autopurge_unregistered_instances_delay**

  *<<**extension**>> (p. 213) Maximum duration for which the **DDSDataWriter** (p. 1589) will maintain information regarding an instance once it has unregistered the instance.*
- struct **DDS_Duration_t autopurge_disposed_instances_delay**

  *<<**extension**>> (p. 213) Maximum duration for which the **DDSDataWriter** (p. 1589) will maintain information regarding an instance once it has disposed the instance.*

9.235.1 Detailed Description

*<<**spec**>> (p. 209)* Controls how a **DDSDataWriter** (p. 1589) handles the lifecycle of the instances (keys) that it is registered to manage.

Entity:

DDSDataWriter (p. 1589)

Properties:

RxO (p. 556) = N/A
Changeable (p. 556) = **YES** (p. 556)

9.235.2 Usage

This policy determines how the **DDSDataWriter** (p. 1589) acts with regards to the lifecycle of the data instances it manages (data instances that have been either explicitly registered with the **DDSDataWriter** (p. 1589) or implicitly registered by directly writing the data).

You may use **FooDataWriter::unregister_instance** (p. 1948) to indicate that the **DDSDataWriter** (p. 1589) no longer wants to send data for a **DDSTopic** (p. 1885).

The behavior controlled by this QoS policy applies on a per instance (key) basis for keyed Topics, so that when a **DDSDataWriter** (p. 1589) unregisters an instance, RTI Connext can automatically also dispose that instance. This is the default behavior.

In many cases where the ownership of a Topic is shared (see **DDS_OwnershipQosPolicy** (p. 1222)), DataWriters may want to relinquish their ownership of a particular instance of the Topic to allow other DataWriters to send updates for the value of that instance regardless of Ownership Strength. In that case, you may only want a DataWriter to unregister an instance without disposing the instance. *Disposing* an instance is a statement that an instance no longer exists. User applications may be coded to trigger on the disposal of instances, thus the ability to unregister without disposing may be useful to properly maintain the semantic of disposal.

9.235.3 Member Data Documentation

9.235.3.1 autodispose_unregistered_instances

```
DDS_Boolean DDS_WriterDataLifecycleQosPolicy::autodispose_unregistered_instances
```

<<***spec***>> *(p. 209)* Boolean flag that controls whether or not a writer will also dispose of an instance when the **DDSDataWriter** (p. 1589) unregisters from the instance by means of the unregister operations.

- **DDS_BOOLEAN_TRUE** (p. 513)
  The **DDSDataWriter** (p. 1589) will dispose of the instance each time it is unregistered. The behavior is identical to explicitly calling one of the `dispose` operations on the instance prior to calling the `unregister` operation.
- **DDS_BOOLEAN_FALSE** (p. 513) (default)
  The **DDSDataWriter** (p. 1589) will not dispose of the instance. The application can still call one of the dispose operations prior to unregistering the instance and dispose of the instance that way.

[default] DDS_BOOLEAN_FALSE (p. 513)

Categories

Immutable (p. 333), **Instance Lifecycle** (p. 286), **Sample Memory Management** (p. 321)

9.235.3.2 autopurge_unregistered_instances_delay

`struct` **`DDS_Duration_t`** `DDS_WriterDataLifecycleQosPolicy::autopurge_unregistered_instances_delay`

<<***extension***>> *(p. 213)* Maximum duration for which the **DDSDataWriter** (p. 1589) will maintain information regarding an instance once it has unregistered the instance.

Determines how long the **DDSDataWriter** (p. 1589) will maintain information regarding an instance that has been unregistered. By default, the **DDSDataWriter** (p. 1589) resources associated with an instance (e.g., the space needed to remember the Instance Key or KeyHash) are released lazily. This means the resources are only reclaimed when the space is needed for another instance because **DDS_ResourceLimitsQosPolicy::max_instances** (p. 1311) is exceeded. This behavior can be changed by setting autopurge_unregistered_instances_delay to a value other than **DDS_DURATION_INFINITE** (p. 522).

After this time elapses, the **DDSDataWriter** (p. 1589) will purge all internal information regarding the instance, including historical samples, even if **DDS_ResourceLimitsQosPolicy::max_instances** (p. 1311) has not been reached.

The purging of unregistered instances can be done based on the source timestamp of the unregister sample or the time where the unregister sample was added to the DataWriter queue by setting the following property to 1 or 0 respectively (default: 0): **dds.data_writer.history.source_timestamp_based_autopurge_instances_delay**.

For durable writer history, autopurge_unregistered_instances_delay supports only the **DDS_DURATION_INFINITE** (p. 522) value.

With **DDS_VOLATILE_DURABILITY_QOS** (p. 598) durability, unregistered instances are removed as soon as all samples for the instance are fully acknowledged, regardless of how this field is set.

[default] DDS_DURATION_INFINITE (p. 522) (disabled) for all **DDSDataWriter** (p. 1589) except for the built-in discovery DataWriters
DDS_DURATION_ZERO (p. 523) for built-in discovery DataWriters (see **DDS_DiscoveryConfigQosPolicy::publication_writer_data_lifecycle** (p. 944), **DDS_DiscoveryConfigQosPolicy::subscription_writer_data_lifecycle** (p. 945) and **DDS_DiscoveryConfigQosPolicy::participant_configuration_writer_data_lifecycle** (p. 958)).

[range] [0, 1 year] or **DDS_DURATION_INFINITE** (p. 522)

Categories

Immutable (p. 333), **Instance Lifecycle** (p. 286), **Sample Memory Management** (p. 321)

9.235.3.3 autopurge_disposed_instances_delay

`struct` **`DDS_Duration_t`** `DDS_WriterDataLifecycleQosPolicy::autopurge_disposed_instances_delay`

<<***extension***>> *(p. 213)* Maximum duration for which the **DDSDataWriter** (p. 1589) will maintain information regarding an instance once it has disposed the instance.

Determines how long the **DDSDataWriter** (p. 1589) will maintain information regarding an instance that has been disposed of. By default, disposing of an instance does not make it eligible to be purged. By setting autopurge_disposed_instances_delay to a value other than **DDS_DURATION_INFINITE** (p. 522), the DataWriter will delete the resources associated with an instance (including historical samples) once the time has elapsed and all matching DataReaders have acknowledged all the samples for this instance including the dispose sample.

The purging of disposed instances can be done based on the source timestamp of the dispose sample or the time when the dispose sample was added to the DataWriter queue by setting the following property to 1 or 0 respectively (default: 0): **dds.data_writer.history.source_timestamp_based_autopurge_instances_delay**.

This QoS value is supported with durable DataWriter queues only for **DDS_DURATION_ZERO** (p. 523) and **DDS_DURATION_INFINITE** (p. 522) values (finite values are not supported).

[default] DDS_DURATION_INFINITE (p. 522) (disabled)

[range] [0, 1 year] or **DDS_DURATION_INFINITE** (p. 522)

Categories

Immutable (p. 333), **Instance Lifecycle** (p. 286), **Sample Memory Management** (p. 321)

9.236 DDS_WstringSeq Struct Reference

Instantiates **`FooSeq`** (p. 1965) < **DDS_Wchar** (p. 514)* >

9.236.1 Detailed Description

Instantiates **`FooSeq`** (p. 1965) < **DDS_Wchar** (p. 514)* >

Instantiates:

<<***generic***>> *(p. 433)* **FooSeq** (p. 1965)

See also

DDS_Wchar (p. 514)

DDS_StringSeq (p. 1364)

FooSeq (p. 1965)

9.237 DDSAsyncWaitSet Class Reference

A class for dispatching **DDSCondition** (p. 1544) objects using separate threads of execution. You can see this class as an extension of a **DDSWaitSet** (p. 1897) that allows asynchronously waiting for the attached **DDSCondition** (p. 1544) objects to trigger and provide a notification by calling **DDSCondition::dispatch** (p. 1546).

Public Member Functions

- virtual **DDS_ReturnCode_t start** ()

  *Initiates the asynchronous wait on this **DDSAsyncWaitSet** (p. 1528).*
- virtual **DDS_ReturnCode_t start_with_completion_token** (**DDSAsyncWaitSetCompletionToken** ∗completion_token)

  *Initiates the asynchronous wait on this **DDSAsyncWaitSet** (p. 1528).*
- virtual **DDS_ReturnCode_t stop** ()

  *Initiates the stop procedure on this **DDSAsyncWaitSet** (p. 1528) that will stop the asynchronous wait.*
- virtual **DDS_ReturnCode_t stop_with_completion_token** (**DDSAsyncWaitSetCompletionToken** ∗completion_token)

  *Initiates the stop procedure on this **DDSAsyncWaitSet** (p. 1528) that will stop the asynchronous wait.*
- virtual **DDS_ReturnCode_t attach_condition** (**DDSCondition** ∗condition)

  *Attaches the specified **DDSCondition** (p. 1544) to this **DDSAsyncWaitSet** (p. 1528).*
- virtual **DDS_ReturnCode_t detach_condition** (**DDSCondition** ∗condition)

  *Deaches the specified **DDSCondition** (p. 1544) from this **DDSAsyncWaitSet** (p. 1528).*
- virtual **DDS_ReturnCode_t attach_condition_with_completion_token** (**DDSCondition** ∗condition, **DDSAsyncWaitSetCompletionToken** ∗completion_token)

  *Attaches the specified **DDSCondition** (p. 1544) to this **DDSAsyncWaitSet** (p. 1528).*
- virtual **DDS_ReturnCode_t detach_condition_with_completion_token** (**DDSCondition** ∗condition, **DDSAsyncWaitSetCompletionToken** ∗completion_token)

  *Detaches the specified **DDSCondition** (p. 1544) from this **DDSAsyncWaitSet** (p. 1528).*
- virtual **DDS_ReturnCode_t unlock_condition** (**DDSCondition** ∗condition)

  *Allows the **DDSCondition** (p. 1544) under dispatch to be available for concurrent dispatch from another thread from the pool.*
- virtual **DDS_ReturnCode_t get_property** (**DDS_AsyncWaitSetProperty_t** &property)

  *Retrieves the **DDS_AsyncWaitSetProperty_t** (p. 802) configuration of the associated **DDSAsyncWaitSet** (p. 1528).*
- virtual **DDS_ReturnCode_t get_conditions** (**DDSConditionSeq** &attached_conditions)

  *Retrieves the list of attached **DDSCondition** (p. 1544) (s).*
- virtual **DDSAsyncWaitSetCompletionToken** ∗ **create_completion_token** ()

  *Creates a new **DDSAsyncWaitSetCompletionToken** (p. 1541).*
- virtual **DDS_ReturnCode_t delete_completion_token** (**DDSAsyncWaitSetCompletionToken** ∗completion↩ _token)

  *Deletes a **DDSAsyncWaitSetCompletionToken** (p. 1541) previously created from this **DDSAsyncWaitSet** (p. 1528).*
- **DDSAsyncWaitSet** ()

  *Creates a **DDSAsyncWaitSet** (p. 1528) with default property.*
- **DDSAsyncWaitSet** (const **DDS_AsyncWaitSetProperty_t** &property)

  *Single-argument constructor that allows creating a a **DDSAsyncWaitSet** (p. 1528) with custom behavior.*
- **DDSAsyncWaitSet** (const **DDS_AsyncWaitSetProperty_t** &property, **DDSAsyncWaitSetListener** ∗listener)

  *Constructor that allows specifying a **DDSAsyncWaitSetListener** (p. 1543).*

- **DDSAsyncWaitSet** (const **DDS_AsyncWaitSetProperty_t** &property, **DDSAsyncWaitSetListener** *listener, **DDSThreadFactory** *thread_factory)

  *Constructor with arguments that allow specifying behavior different than the default one, including specifying a **DDSThreadFactory** (p. 1883) for the creation and deletion of the threads within the thread pool.*
- virtual ~**DDSAsyncWaitSet** ()

  *Deletes a **DDSAsyncWaitSet** (p. 1528).*

Static Public Attributes

- static **DDSAsyncWaitSetCompletionToken** *const **COMPLETION_TOKEN_USE_IMPLICIT_AND_WAIT**

  *For the operations that allow an **DDSAsyncWaitSetCompletionToken** (p. 1541), this sentinel can be provided to indicate an **DDSAsyncWaitSet** (p. 1528) to use the implicit completion token and wait on it for request completion.*
- static **DDSAsyncWaitSetCompletionToken** *const **COMPLETION_TOKEN_IGNORE**

  *For the operations that allow an **DDSAsyncWaitSetCompletionToken** (p. 1541), this sentinel can be provided to indicate an **DDSAsyncWaitSet** (p. 1528) to perform the action associating a 'null' completion token.*

9.237.1 Detailed Description

A class for dispatching **DDSCondition** (p. 1544) objects using separate threads of execution. You can see this class as an extension of a **DDSWaitSet** (p. 1897) that allows asynchronously waiting for the attached **DDSCondition** (p. 1544) objects to trigger and provide a notification by calling **DDSCondition::dispatch** (p. 1546).

DDSAsyncWaitSet (p. 1528) provides a proactive model to process application events through **DDSCondition** (p. 1544) objects. **DDSAsyncWaitSet** (p. 1528) owns a pool of threads to asynchronously wait for the attached **DDSCondition** (p. 1544) objects to trigger and dispatch them upon wakeup. The asynchronous behavior is the main key different with regards to the **DDSWaitSet** (p. 1897).

The class diagram and its collaborators is shown below:

9.237.2 AsyncWaitSet Thread Orchestration

DDSAsyncWaitSet (p. 1528) internally applies a leader-follower pattern for the orchestration of the thread pool. Once a **DDSAsyncWaitSet** (p. 1528) starts, it will create the thread pool of `M` threads from which only one thread will become the Leader thread, and remaining threads will become the Followers, where:

- The `Leader` thread is the one waiting for the attached **DDSCondition** (p. 1544) to trigger. Remaining threads in the pool, if any, are either idle awaiting to become the leader or busy while processing active **DDSCondition** (p. 1544).
- Upon wait wakeup, the `Leader` thread resigns its leader status to become a `Processor` thread and dispatch the next active **DDSCondition** (p. 1544) through the **DDSCondition::dispatch** (p. 1546) operation.
- One of the `Follower` threads wakes up and becomes the new leader to resume the wait for **DDSCondition** (p. 1544).

"Thread orchestration in a ::DDSAsyncWaitSet"

This behavior implies the following considerations:

- Only one thread a time can wait for the attached **DDSCondition** (p. 1544) objects to trigger. From a pool of `M` threads, only one is the leader, `P` are processing active **DDSCondition** (p. 1544), and `F` are idle followers.
- A thread can dispatch only one active **DDSCondition** (p. 1544) at a time.
- **DDSAsyncWaitSet** (p. 1528) efficiently distributes threads to dispatch **DDSCondition** (p. 1544) objects on demand. This avoids underutilizing a thread if **DDSCondition** (p. 1544) objects do not trigger or trigger unfrequently.
- At a given time, all the threads in the pool could be in processing state. In this situation, the **DDSAsyncWaitSet** (p. 1528) is not able to wait for more **DDSCondition** (p. 1544) objects until one thread becomes the leader.

DDSAsyncWaitSet (p. 1528) has a built-in dispatcher that guarantees fairness and avoids starvation of **DDSCondition** (p. 1544) objects. By applying a round-robin distribution policy, each attached and active **DDSCondition** (p. 1544) is dispatched within a finite period of time, assuming the **DDSConditionHandler** (p. 1546) always return control after the **DDSCondition::dispatch** (p. 1546) operation.

9.237.3 AsyncWaitSet Thread Safety

A key aspect of the **DDSAsyncWaitSet** (p. 1528) is the thread safety. **DDSAsyncWaitSet** (p. 1528) interface is thread safe, so you can concurrently call any operation on the **DDSAsyncWaitSet** (p. 1528) object from multiple threads in your application.

Furthermore, **DDSAsyncWaitSet** (p. 1528) also safely interacts with its own thread pool. Internally, the **DDSAsyncWaitSet** (p. 1528) applies the asynchronous completion token pattern to perform activities that involve synchronization with the thread pool.

For instance to detach a **DDSCondition** (p. 1544), the **DDSAsyncWaitSet** (p. 1528) generates an internal request to its thread pool to process it. As soon as the detachment completes, the thread pool provides the notification through an associated completion token.

Note

The asynchronous completion token behavior only takes place if the **DDSAsyncWaitSet** (p. 1528) is started. Otherwise the internal request will be directly executed by the calling thread.

For a finer control on this behavior, each **DDSAsyncWaitSet** (p. 1528) operation where this applies comes in two flavors:

- `Default:` the operation hides all the details of the completion token and returns after the operation completes. Operations of this kind internally use an implicit **DDSAsyncWaitSetCompletionToken** (p. 1541). The **DDSAsyncWaitSet** (p. 1528) creates and reuses **DDSAsyncWaitSetCompletionToken** (p. 1541) objects as needed. This is the recommended flavor unless your application has special resource needs.
- `With completion token:` An overloaded version of the default one that also receives an **DDSAsyncWaitSetCompletionToken** (p. 1541) object on which you can wait on at any time for the actual operation to complete. This flavor is available to assist applications with resource constraints and that want more control on the interaction with the thread pool of the **DDSAsyncWaitSet** (p. 1528).

9.237.3.1 Condition Locking

DDSAsyncWaitSet (p. 1528) incorporates a safety mechanism that prevents calling **DDSCondition::dispatch** (p. 1546) concurrently. **DDSAsyncWaitSet** (p. 1528) locks the **DDSCondition** (p. 1544) while a processor thread is dispatching it so no other thread within the pool can dispatch it again.

This mechanism ensures not only unexpected concurrent dispatch of a **DDSCondition** (p. 1544) but also spurious thread activity. Because it is responsibility of your application to reset the Condition trigger, there is a period of time in which the dispatched condition may remain active, causing the **DDSAsyncWaitSet** (p. 1528) to enter in a continous immediate wakeup from the wait. This behavior typically leads to thread hogging and high CPU usage.

Nevertheless, your application may still want to receive concurrent and controlled dispatch notifications. **DDSAsync↩ WaitSet** (p. 1528) will still allows you to unlock a **DDSCondition** (p. 1544) so any other available thread can dispatch the same condition concurrently while preventing the above mentioned problems. You can achive this by calling **DDSAsyncWaitSet::unlock_condition** (p. 1539) on the Condition being dispatched within the dispatch callback. Note that the AsyncWaitSet locks a Condition each time it dispatches it. Hence you need to unlock the Condition each time you want to enable a concurrent dispatch.

9.237.4 AsyncWaitSet Events and Resources

Besides **DDSCondition** (p. 1544) processing, you can listen to other kind of internal events related to the **DDSAsync↩ WaitSet** (p. 1528) and its thread pool by means of the **DDSAsyncWaitSetListener** (p. 1543).

DDSAsyncWaitSet (p. 1528) exposes operations to start and stop the asynchronous wait, which involves the creation and deletion of the thread pool respectively.

DDSAsyncWaitSet (p. 1528) relies on thread-specific storage to provide the described functionality. Each application thread that calls an operation on a **DDSAsyncWaitSet** (p. 1528) will generate resources that will be associated with such thread. You can free these resources upon thread termination by calling **DDSDomainParticipantFactory::unregister↩ _thread** (p. 1716).

MT Safety:

Safe.

See also

DDSWaitSet (p. 1897)

DDSCondition (p. 1544)

DDSAsyncWaitSetListener (p. 1543)

DDSAsyncWaitSetCompletionToken (p. 1541).

DDS_AsyncWaitSetProperty_t (p. 802)

9.237.5 Constructor & Destructor Documentation

9.237.5.1 DDSAsyncWaitSet() [1/4]

```
DDSAsyncWaitSet::DDSAsyncWaitSet ( )
```

Creates a **DDSAsyncWaitSet** (p. 1528) with default property.

9.237.5.2 DDSAsyncWaitSet() [2/4]

```
DDSAsyncWaitSet::DDSAsyncWaitSet (
            const  DDS_AsyncWaitSetProperty_t & property )
```

Single-argument constructor that allows creating a a **DDSAsyncWaitSet** (p. 1528) with custom behavior.

You can provide **DDS_ASYNC_WAITSET_PROPERTY_DEFAULT** (p. 489) as `property` to create an **DDSAsync↩ WaitSet** (p. 1528) with default behavior.

The **DDSAsyncWaitSet** (p. 1528) is created with no listener installed.

Parameters

property	<<***in***>> *(*p. *433)* configuration **DDS_AsyncWaitSetProperty_t** (p. 802)

9.237.5.3 DDSAsyncWaitSet() [3/4]

```
DDSAsyncWaitSet::DDSAsyncWaitSet (
            const  DDS_AsyncWaitSetProperty_t & property,
             DDSAsyncWaitSetListener * listener )
```

Constructor that allows specifying a **DDSAsyncWaitSetListener** (p. 1543).

Creates a new **DDSAsyncWaitSet** (p. 1528) with the specified property **DDS_AsyncWaitSetProperty_t** (p. 802) and **DDSAsyncWaitSetListener** (p. 1543).

Parameters

property	<<***in***>> *(*p. *433)* configuration **DDS_AsyncWaitSetProperty_t** (p. 802)
listener	<<***in***>> *(*p. *433)* the **DDSAsyncWaitSetListener** (p. 1543). Cannot be NULL.

9.237.5.4 DDSAsyncWaitSet() [4/4]

```
DDSAsyncWaitSet::DDSAsyncWaitSet (
            const  DDS_AsyncWaitSetProperty_t & property,
             DDSAsyncWaitSetListener * listener,
             DDSThreadFactory * thread_factory )
```

Constructor with arguments that allow specifying behavior different than the default one, including specifying a **DDSThreadFactory** (p. 1883) for the creation and deletion of the threads within the thread pool.

This operation extends **DDSAsyncWaitSet::DDSAsyncWaitSet()** (p. 1531) by allowing to provide a **DDSThread↩ Factory** (p. 1883)

Parameters

property	<<***in***>> *(p. 433)* configuration **DDS_AsyncWaitSetProperty_t** (p. 802)
listener	<<***in***>> *(p. 433)* the **DDSAsyncWaitSetListener** (p. 1543). Cannot be NULL.
thread_factory	<<***in***>> *(p. 433)* **DDSThreadFactory** (p. 1883) for the creation and deletion of threads.

9.237.5.5 ∼DDSAsyncWaitSet()

```
virtual DDSAsyncWaitSet::∼DDSAsyncWaitSet ( )  [virtual]
```

Deletes a **DDSAsyncWaitSet** (p. 1528).

If the **DDSAsyncWaitSet** (p. 1528) is started, this operation will initiate the stop procedure and block until it completes.

The deletion will fail if there are outstanding **DDSAsyncWaitSetCompletionToken** (p. 1541) that have not been deleted.

Any outstanding **DDSAsyncWaitSet** (p. 1528) must be deleted before finalizing the **DDSDomainParticipantFactory** (p. 1694). Otherwise undefined behavior may occur.

See also

DDSAsyncWaitSet::DDSAsyncWaitSet() (p. 1531)

9.237.6 Member Function Documentation

9.237.6.1 start()

```
virtual  DDS_ReturnCode_t DDSAsyncWaitSet::start ( )  [virtual]
```

Initiates the asynchronous wait on this **DDSAsyncWaitSet** (p. 1528).

This operation is equivalent to calling **DDSAsyncWaitSet::start_with_completion_token** (p. 1534) providing **DDSAsyncWaitSet::COMPLETION_TOKEN_USE_IMPLICIT_AND_WAIT** (p. 489) as a `completion_token`.

This operation blocks until the start request completes. Upon successful return, it is guaranteed that this **DDSAsync↩ WaitSet** (p. 1528) has initiated the asynchronous wait and dispatch.

Exceptions

One	of the **Standard Return Codes** (p. 532)

See also

DDSAsyncWaitSet::start_with_completion_token (p. 1534)

DDSAsyncWaitSet::stop (p. 1535)

9.237.6.2 start_with_completion_token()

```
virtual DDS_ReturnCode_t DDSAsyncWaitSet::start_with_completion_token (
            DDSAsyncWaitSetCompletionToken * completion_token ) [virtual]
```

Initiates the asynchronous wait on this **DDSAsyncWaitSet** (p. 1528).

If this operation succeeds, a start request has been scheduled and your application can use the provided `completion_token` to wait for this **DDSAsyncWaitSet** (p. 1528) to process the request. If the **DDSAsync↩ WaitSetCompletionToken::wait** (p. 1542) operation returns successfully, it is guranteed that the thread pool has been created and the leader thread is waiting for the attached **DDSCondition** (p. 1544) to trigger.

Once this **DDSAsyncWaitSet** (p. 1528) is started, attached **DDSCondition** (p. 1544) will be dispatched through the **DDSCondition::dispatch** (p. 1546) operation when they trigger.

The start procedure causes the **DDSAsyncWaitSet** (p. 1528) to spawn all the threads within the thread pool, which involves the underlying operating system to allocate the associated thread stack and context for each thread. If a **DDSAsyncWaitSetListener** (p. 1543) is installed, this **DDSAsyncWaitSet** (p. 1528) will sequentially invoke the **DDSAsyncWaitSetListener::on_thread_spawned** (p. 1543) once per spwaned thread.

A **DDSAsyncWaitSet** (p. 1528) can be restarted after a stop. If this **DDSAsyncWaitSet** (p. 1528) is already started, this operation will return immediately with success, and waiting on the `completion_token` will also return immediately with success.

Parameters

completion_token	<<***inout***>> *(p. 433)* a valid **DDSAsyncWaitSetCompletionToken** (p. 1541) instance that can be used by your application to wait for the start request to complete. You can provide one of the special sentinels **DDSAsyncWaitSet::COMPLETION_TOKEN_USE_IMPLICIT_AND_WAIT** (p. 489) and **DDSAsyncWaitSet::COMPLETION_TOKEN_IGNORE** (p. 489).

Exceptions

One	of the **Standard Return Codes** (p. 532)

See also

DDSAsyncWaitSet::start (p. 1533)

DDSAsyncWaitSet::stop_with_completion_token (p. 1535)

DDSWaitSet::wait (p. 1901)

9.237.6.3 stop()

```
virtual DDS_ReturnCode_t DDSAsyncWaitSet::stop ( ) [virtual]
```

Initiates the stop procedure on this **DDSAsyncWaitSet** (p. 1528) that will stop the asynchronous wait.

This operation is equivalent to calling **DDSAsyncWaitSet::stop_with_completion_token** (p. 1535) providing **DDSAsyncWaitSet::COMPLETION_TOKEN_USE_IMPLICIT_AND_WAIT** (p. 489) as a `completion_token`.

This operation will block until the stop request completes. Upon successful return, it is guaranteed that this **DDSAsync↩ WaitSet** (p. 1528) stopped the asynchronous wait and dispatch.

Exceptions

One	of the **Standard Return Codes** (p. 532)

See also

DDSAsyncWaitSet::start_with_completion_token (p. 1534)

DDSAsyncWaitSet::stop (p. 1535)

9.237.6.4 stop_with_completion_token()

```
virtual DDS_ReturnCode_t DDSAsyncWaitSet::stop_with_completion_token (
            DDSAsyncWaitSetCompletionToken * completion_token ) [virtual]
```

Initiates the stop procedure on this **DDSAsyncWaitSet** (p. 1528) that will stop the asynchronous wait.

If this operation succeeds, a stop request has been scheduled and your application can use the provided `completion_token` to wait for this **DDSAsyncWaitSet** (p. 1528) to process the request. If the **DDSAsync↩ WaitSetCompletionToken::wait** (p. 1542) operation returns successfully, it is guranteed that the thread pool has been deleted and this **DDSAsyncWaitSet** (p. 1528) no longer process any of the attached **DDSCondition** (p. 1544) objects.

Once this **DDSAsyncWaitSet** (p. 1528) is stopped, the **DDSCondition::dispatch** (p. 1546) will no longer be called on any of the attached **DDSCondition** (p. 1544), no matter what their trigger value is.

The stop procedure causes the **DDSAsyncWaitSet** (p. 1528) to delete all the threads within the thread pool, which involves the underlying operating system to release the associated thread stack and context of each thread. If a **DDSAsyncWaitSetListener** (p. 1543) is installed, this **DDSAsyncWaitSet** (p. 1528) will sequentially invoke the **DDSAsyncWaitSetListener::on_thread_deleted** (p. 1543) once per deleted thread.

If this **DDSAsyncWaitSet** (p. 1528) is already stopped, this operation will return immediately with success, and waiting on the `completion_token` will also return immediately with success.

Parameters

completion_token	<<***inout***>> *(p. 433)* a valid **DDSAsyncWaitSetCompletionToken** (p. 1541) instance that can be used by your application to wait for the stop request to complete. You can provide one of the special sentinels **DDSAsyncWaitSet::COMPLETION_TOKEN_USE_IMPLICIT_AND_WAIT** (p. 489) and **DDSAsyncWaitSet::COMPLETION_TOKEN_IGNORE** (p. 489).

Exceptions

One	of the **Standard Return Codes** (p. 532)

See also

DDSAsyncWaitSet::stop (p. 1535)

DDSAsyncWaitSet::start_with_completion_token (p. 1534)

9.237.6.5 attach_condition()

```
virtual DDS_ReturnCode_t DDSAsyncWaitSet::attach_condition (
            DDSCondition * condition ) [virtual]
```

Attaches the specified **DDSCondition** (p. 1544) to this **DDSAsyncWaitSet** (p. 1528).

This operation is equivalent to calling **DDSAsyncWaitSet::attach_condition_with_completion_token** (p. 1537) providing **DDSAsyncWaitSet::COMPLETION_TOKEN_USE_IMPLICIT_AND_WAIT** (p. 489) as a `completion_token`.

This operation will block until the attach request completes. Upon successful return, it is guaranteed that the specified **DDSCondition** (p. 1544) is attached.

If the same **DDSCondition** (p. 1544) is attached multiple times, only the first attachment has any effect. Subsequent attempts to attach the same **DDSCondition** (p. 1544) are silently ignored. As a result, the **DDSCondition** (p. 1544) is only considered attached once and must only be detached once.

Parameters

condition	<<***in***>> *(p. 433)* **DDSCondition** (p. 1544) to be attached.

Exceptions

One	of the **Standard Return Codes** (p. 532)

See also

DDSAsyncWaitSet::attach_condition_with_completion_token (p. 1537)

9.237.6.6 detach_condition()

```
virtual  DDS_ReturnCode_t DDSAsyncWaitSet::detach_condition (
              DDSCondition * condition )  [virtual]
```

Deaches the specified **DDSCondition** (p. 1544) from this **DDSAsyncWaitSet** (p. 1528).

This operation is equivalent to call **DDSAsyncWaitSet::detach_condition_with_completion_token** (p. 1538) providing **DDSAsyncWaitSet::COMPLETION_TOKEN_USE_IMPLICIT_AND_WAIT** (p. 489) as a `completion_token`.

This operation blocks until the detach request completes. Upon successful return, it is guaranteed that the specified **DDSCondition** (p. 1544) is detached.

Parameters

condition	<<***in***>> *(p. 433)* **DDSCondition** (p. 1544) to be detached.

Exceptions

One	of the **Standard Return Codes** (p. 532)

See also

DDSAsyncWaitSet::detach_condition_with_completion_token (p. 1538)

9.237.6.7 attach_condition_with_completion_token()

```
virtual  DDS_ReturnCode_t DDSAsyncWaitSet::attach_condition_with_completion_token (
              DDSCondition * condition,
              DDSAsyncWaitSetCompletionToken * completion_token )  [virtual]
```

Attaches the specified **DDSCondition** (p. 1544) to this **DDSAsyncWaitSet** (p. 1528).

If this operation succeeds, an attach request has been scheduled and your application can use the output parameter `completion_token` to wait for this **DDSAsyncWaitSet** (p. 1528) to process the request. **DDSAsyncWaitSet↩ CompletionToken::wait** (p. 1542) operation returns successfully, it is guaranteed that the **DDSCondition** (p. 1544) is attached to this **DDSAsyncWaitSet** (p. 1528).

Once the **DDSCondition** (p. 1544) is attached, its trigger value may cause the leader thread of the **DDSAsyncWaitSet** (p. 1528) to wake up call the **DDSCondition::dispatch** (p. 1546) operation.

DDSCondition (p. 1544) may be attached at any time independently of the state of the **DDSAsyncWaitSet** (p. 1528).

Parameters

condition	*<<**in**>> (p. 433)* **DDSCondition** (p. 1544) to be attached.
completion_token	*<<**inout**>> (p. 433)* a valid **DDSAsyncWaitSetCompletionToken** (p. 1541) instance that can be used by your application to wait for the attach request to complete. You can provide one of the special sentinels **DDSAsyncWaitSet::COMPLETION_TOKEN_USE_IMPLICIT_AND_WAIT** (p. 489) and **DDSAsyncWaitSet::COMPLETION_TOKEN_IGNORE** (p. 489).

Exceptions

One	of the **Standard Return Codes** (p. 532)

See also

DDSAsyncWaitSet::attach_condition (p. 1536)

DDSAsyncWaitSet::detach_condition_with_completion_token (p. 1538)

DDSAsyncWaitSetCompletionToken::wait (p. 1542)

9.237.6.8 detach_condition_with_completion_token()

```
virtual DDS_ReturnCode_t DDSAsyncWaitSet::detach_condition_with_completion_token (
            DDSCondition * condition,
            DDSAsyncWaitSetCompletionToken * completion_token ) [virtual]
```

Detaches the specified **DDSCondition** (p. 1544) from this **DDSAsyncWaitSet** (p. 1528).

If this operation succeeds, a detach request has been scheduled and your application can use the provided `completion_token` to wait for this **DDSAsyncWaitSet** (p. 1528) to process the request. If the **DDSAsyncWaitSetCompletionToken::wait** (p. 1542) operation returns successfully, it is guaranteed that the **DDSCondition** (p. 1544) is detached from this **DDSAsyncWaitSet** (p. 1528).

Once the **DDSCondition** (p. 1544) is detached, it is guaranteed that the **DDSAsyncWaitSet** (p. 1528) will no longer process it so it is safe for your application to release any resources associated with the detached **DDSCondition** (p. 1544).

DDSCondition (p. 1544) may be detached at any time independently of the state of the **DDSAsyncWaitSet** (p. 1528).

If the same **DDSCondition** (p. 1544) had been previously attached multiple times, only a single detach is required; the **DDSCondition** (p. 1544) is considered attached only once.

Parameters

condition	*<<**in**>> (p. 433)* **DDSCondition** (p. 1544) to be detached.
completion_token	*<<**inout**>> (p. 433)* a valid **DDSAsyncWaitSetCompletionToken** (p. 1541) instance that can be used by your application to wait for the detach request to complete. You can provide one of the special sentinels
	DDSAsyncWaitSet::COMPLETION_TOKEN_USE_IMPLICIT_AND_WAIT (p. 489) and **DDSAsyncWaitSet::COMPLETION_TOKEN_IGNORE** (p. 489).

Exceptions

One	of the **Standard Return Codes** (p. 532)

See also

DDSAsyncWaitSet::detach_condition (p. 1537)

DDSAsyncWaitSet::attach_condition_with_completion_token (p. 1537)

DDSAsyncWaitSetCompletionToken::wait (p. 1542)

9.237.6.9 unlock_condition()

```
virtual DDS_ReturnCode_t DDSAsyncWaitSet::unlock_condition (
            DDSCondition * condition ) [virtual]
```

Allows the **DDSCondition** (p. 1544) under dispatch to be available for concurrent dispatch from another thread from the pool.

This operation can be called from the dispatch callback of the **DDSCondition** (p. 1544) this **DDSAsyncWaitSet** (p. 1528) is dispatching. After succesfully calling this operation, if the **DDSCondition** (p. 1544) becomes active this **DDSAsync↩ WaitSet** (p. 1528) is allowed to dispatch it again from any available thread from the pool.

You may call this operation any time you need the same **DDSCondition** (p. 1544) to be dispatched concurrently.

This operation will fail with **DDS_RETCODE_PRECONDITION_NOT_MET** (p. 532) if you call it from a different context than the dispatch callback or on a different **DDSCondition** (p. 1544).

Exceptions

One	of the **Standard Return Codes** (p. 532)

9.237.6.10 get_property()

```
virtual DDS_ReturnCode_t DDSAsyncWaitSet::get_property (
            DDS_AsyncWaitSetProperty_t & property ) [virtual]
```

Retrieves the **DDS_AsyncWaitSetProperty_t** (p. 802) configuration of the associated **DDSAsyncWaitSet** (p. 1528).

Parameters

property	<<***out***>> *(p. 433)*

Exceptions

One	of the **Standard Return Codes** (p. 532)

9.237.6.11 get_conditions()

```
virtual DDS_ReturnCode_t DDSAsyncWaitSet::get_conditions (
            DDSConditionSeq & attached_conditions ) [virtual]
```

Retrieves the list of attached **DDSCondition** (p. 1544) (s).

Parameters

attached_conditions	*<<**inout**>> (p. 433)* a **DDSConditionSeq** (p. 1547) object where the list of attached conditions will be returned.

Exceptions

One	of the **Standard Return Codes** (p. 532)

See also

DDSAsyncWaitSet::attach_condition (p. 1536)

DDSAsyncWaitSet::detach_condition (p. 1537)

9.237.6.12 create_completion_token()

```
virtual DDSAsyncWaitSetCompletionToken * DDSAsyncWaitSet::create_completion_token ( ) [virtual]
```

Creates a new **DDSAsyncWaitSetCompletionToken** (p. 1541).

All the created **DDSAsyncWaitSetCompletionToken** (p. 1541) must be deleted by calling **DDSAsyncWaitSet↩ ::delete_completion_token** (p. 1540).

Returns

A new **DDSAsyncWaitSetCompletionToken** (p. 1541) or NULL on error.

See also

DDSAsyncWaitSet::delete_completion_token (p. 1540)

DDSAsyncWaitSet::∼DDSAsyncWaitSet() (p. 1533)

9.237.6.13 delete_completion_token()

```
virtual  DDS_ReturnCode_t DDSAsyncWaitSet::delete_completion_token (
             DDSAsyncWaitSetCompletionToken * completion_token )  [virtual]
```

Deletes a **DDSAsyncWaitSetCompletionToken** (p. 1541) previously created from this **DDSAsyncWaitSet** (p. 1528).

This operation will fail the if the specified `completion_token` was not created from this **DDSAsyncWaitSet** (p. 1528).

This operation will fail if the specified `completion_token` is associated with a request that has not completed yet.

Parameters

completion_token	<<***inout***>> *(p. 433)* a valid **DDSAsyncWaitSetCompletionToken** (p. 1541) created from this **DDSAsyncWaitSet** (p. 1528).

Exceptions

One	of the **Standard Return Codes** (p. 532)

See also

DDSAsyncWaitSet::create_completion_token (p. 1540)

DDSAsyncWaitSet::~DDSAsyncWaitSet() (p. 1533)

9.238 DDSAsyncWaitSetCompletionToken Class Reference

<<***interface***>> *(p. 433)* Implementation of the completion token role element of the asynchronous completion token pattern that is part of the **DDSAsyncWaitSet** (p. 1528) behavior.

Public Member Functions

- virtual **DDS_ReturnCode_t wait** (const **DDS_Duration_t** &max_wait)=0

  *Waits for the request associated to an operation made on the **DDSAsyncWaitSet** (p. 1528) to complete.*

9.238.1 Detailed Description

<<***interface***>> *(p. 433)* Implementation of the completion token role element of the asynchronous completion token pattern that is part of the **DDSAsyncWaitSet** (p. 1528) behavior.

A **DDSAsyncWaitSetCompletionToken** (p. 1541) can be in one of the following states:

- READY: The completion token can be used to associate a new request. Calling **DDSAsyncWaitSet↩ CompletionToken::wait** (p. 1542) on a ready completion token will return immediately with success. A ready completion token can only transition to the queued state.
- QUEUED: The completion token has an associated request that is pending processing. Calling **DDSAsync↩ WaitSetCompletionToken::wait** (p. 1542) on a queued completion token will block until the request completes or times out. A queued completion token can only transition to the processed state.
- PROCESS: The completion token has an associated request that has been processed but the application did not call **DDSAsyncWaitSetCompletionToken::wait** (p. 1542) yet. Calling **DDSAsyncWaitSetCompletionToken↩ ::wait** (p. 1542) on a processed completion token will return immediately with the return code result of processing the associated request. A processed completion token can transition to both ready or queued states.

9.238.2 AsyncWaitSetCompletionToken management

The same **DDSAsyncWaitSetCompletionToken** (p. 1541) instance can be reused reused multiple times to associate a request and wait for its completion. Reusing is allowed only if the completion token is either in READY or PROCESSED state. Otherwise the **DDSAsyncWaitSet** (p. 1528) operation that associates the completion token will fail with **DDS_↩ RETCODE_PRECONDITION_NOT_MET** (p. 532).

The completion token functionality can be viewed as a **DDSAsyncWaitSet** (p. 1528) internal detail from which your application should not need to know. In general, it is recommended to use the default flavor of **DDSAsyncWaitSet** (p. 1528) operations that handle the internals of the completion tokens for you.

Nevertheless, if a completion token represents an expensive resource in your environment, your application may want to have full control of how and when completion tokens are created. It's for these reasons why is exposed as a public collaborator of the **DDSAsyncWaitSet** (p. 1528).

MT Safety:

Safe.

See also

DDSAsyncWaitSet (p. 1528)

9.238.3 Member Function Documentation

9.238.3.1 wait()

```
virtual DDS_ReturnCode_t DDSAsyncWaitSetCompletionToken::wait (
            const DDS_Duration_t & max_wait ) [pure virtual]
```

Waits for the request associated to an operation made on the **DDSAsyncWaitSet** (p. 1528) to complete.

This operation will block the calling thread for a maximum amount of time specified by max_wait until the **DDSAsync↩ WaitSet** (p. 1528) request associated with this completion token completes.

If there is no timeout, upon return it is guaranteed that the request associated with this token completed. This operation may fail due to an error during the wait or while processing the associated request.

If this operation is called from within the context of one of the threads that comprise the thread pool of the **DDSAsync↩ WaitSet** (p. 1528), it will fail with **DDS_RETCODE_PRECONDITION_NOT_MET** (p. 532).

If the operation failed with **DDS_RETCODE_TIMEOUT** (p. 533) your application can wait again on this completion token.

Parameters

max_wait	*<<**in**>> (p. 433)* Cannot be NULL. Maximum time to wait for the request associated to this completion token to complete before timeout.

Exceptions

One	of the **Standard Return Codes** (p. 532)

See also

DDSAsyncWaitSet (p. 1528)

9.239 DDSAsyncWaitSetListener Class Reference

*<<**interface**>> (p. 433)* Listener for receiving event notifications related to the thread pool of the **DDSAsyncWaitSet** (p. 1528).

Public Member Functions

- virtual void **on_thread_spawned** (RTI_UINT64 thread_id)

  *Handles the spawning of each thread comprising the thread pool of the **DDSAsyncWaitSet** (p. 1528).*
- virtual void **on_thread_deleted** (RTI_UINT64 thread_id)

  *Handles the deletion of each thread comprising the thread pool of the **DDSAsyncWaitSet** (p. 1528).*
- virtual void **on_wait_timeout** (RTI_UINT64 thread_id)

  *Handles the wait timeout generated by the leader thread of the **DDSAsyncWaitSet** (p. 1528).*

9.239.1 Detailed Description

*<<**interface**>> (p. 433)* Listener for receiving event notifications related to the thread pool of the **DDSAsyncWaitSet** (p. 1528).

9.239.2 Member Function Documentation

9.239.2.1 on_thread_spawned()

```
virtual void DDSAsyncWaitSetListener::on_thread_spawned (
            RTI_UINT64 thread_id ) [virtual]
```

Handles the spawning of each thread comprising the thread pool of the **DDSAsyncWaitSet** (p. 1528).

9.239.2.2 on_thread_deleted()

```
virtual void DDSAsyncWaitSetListener::on_thread_deleted (
            RTI_UINT64 thread_id )  [virtual]
```

Handles the deletion of each thread comprising the thread pool of the **DDSAsyncWaitSet** (p. 1528).

9.239.2.3 on_wait_timeout()

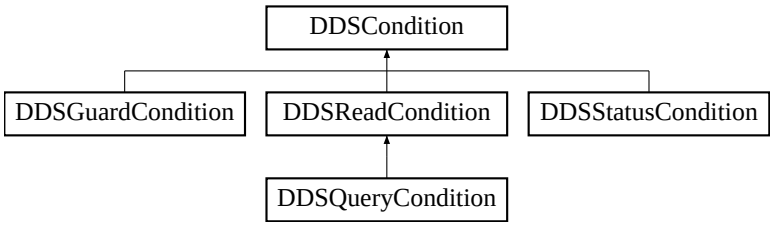
```
virtual void DDSAsyncWaitSetListener::on_wait_timeout (
            RTI_UINT64 thread_id )  [virtual]
```

Handles the wait timeout generated by the leader thread of the **DDSAsyncWaitSet** (p. 1528).

9.240 DDSCondition Class Reference

<<***interface***>> *(p. 433)* Root class for all the conditions that may be attached to a **DDSWaitSet** (p. 1897).

Inheritance diagram for DDSCondition:



Public Member Functions

- virtual **DDS_Boolean get_trigger_value** ()=0

  Retrieve the `trigger_value`*.*
- virtual **DDS_ReturnCode_t set_handler** (**DDSConditionHandler** *handler)

  <<***extension***>> *(p. 432) Registers a* ***DDSConditionHandler*** *(p. 1546) in this* ***DDSCondition*** *(p. 1544).*
- virtual **DDSConditionHandler** * **get_handler** ()

  <<***extension***>> *(p. 432) Returns the registered* ***DDSConditionHandler*** *(p. 1546).*
- virtual void **dispatch** ()

  <<***extension***>> *(p. 432) Calls* ***DDSConditionHandler::on_condition_triggered*** *(p. 1547) of the registered* ***DDSConditionHandler*** *(p. 1546).*

9.240.1 Detailed Description

<<***interface***>> *(p. 433)* Root class for all the conditions that may be attached to a **DDSWaitSet** (p. 1897).

This basic class is specialised in three classes:

DDSGuardCondition (p. 1739), **DDSStatusCondition** (p. 1846), and **DDSReadCondition** (p. 1842).

A **DDSCondition** (p. 1544) has a `trigger_value` that can be **DDS_BOOLEAN_TRUE** (p. 513) or **DDS_BOOLEAN_FALSE** (p. 513) and is set automatically by RTI Connext.

See also

DDSWaitSet (p. 1897)

9.240.2 Member Function Documentation

9.240.2.1 get_trigger_value()

```
virtual DDS_Boolean DDSCondition::get_trigger_value ( ) [pure virtual]
```

Retrieve the `trigger_value`.

Returns

the trigger value.

Implemented in **DDSGuardCondition** (p. 1741).

9.240.2.2 set_handler()

```
virtual DDS_ReturnCode_t DDSCondition::set_handler (
            DDSConditionHandler * handler ) [virtual]
```

<<***extension***>> *(p. 432)* Registers a **DDSConditionHandler** (p. 1546) in this **DDSCondition** (p. 1544).

This operation replaces any existing registered handler. If there is any resources associated with an existing registered handler that need to be released, you may first call **DDSCondition::get_handler** (p. 1546) to retrieve the handler.

MT Safety:

It is not safe to call **DDSCondition::set_handler** (p. 1545), **DDSCondition::get_handler** (p. 1546) or **DDSCondition::dispatch** (p. 1546) concurrently.

Parameters

handler	The **DDSConditionHandler** (p. 1546) to be called by **DDSCondition::dispatch** (p. 1546). If this parameter is null, an no-op handler implemenation will be set.

9.240.2.3 get_handler()

```
virtual DDSConditionHandler * DDSCondition::get_handler ( ) [virtual]
```

<<***extension***>> *(p. 432)* Returns the registered **DDSConditionHandler** (p. 1546).

If no **DDSConditionHandler** (p. 1546) is registered, this operation returns a no-op **DDSConditionHandler** (p. 1546) implementation.

9.240.2.4 dispatch()

```
virtual void DDSCondition::dispatch ( ) [virtual]
```

<<***extension***>> *(p. 432)* Calls **DDSConditionHandler::on_condition_triggered** (p. 1547) of the registered **DDSConditionHandler** (p. 1546).

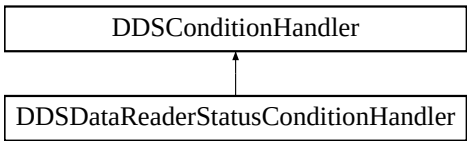
If the trigger value is true, calling this operation will call the registered **DDSConditionHandler** (p. 1546).

If no **DDSConditionHandler** (p. 1546) is registered, this operation is a no-op.

9.241 DDSConditionHandler Class Reference

<<***extension***>> *(p. 432)* <<***interface***>> *(p. 433)* Handler called by the **DDSCondition::dispatch** (p. 1546).

Inheritance diagram for DDSConditionHandler:



Public Member Functions

- virtual void **on_condition_triggered** (**DDSCondition** *condition)=0

  *Handles the dispatch of a **DDSCondition** (p. 1544).*

9.241.1 Detailed Description

<<***extension***>> *(*p. *432)* <<***interface***>> *(*p. *433)* Handler called by the **DDSCondition::dispatch** (p. 1546).

Direct known implementations: **DDSDataReaderStatusConditionHandler** (p. 1586)

9.241.2 Member Function Documentation

9.241.2.1 on_condition_triggered()

```
virtual void DDSConditionHandler::on_condition_triggered (
              DDSCondition * condition )  [pure virtual]
```

Handles the dispatch of a **DDSCondition** (p. 1544).

This callback is called by **DDSCondition::dispatch** (p. 1546).

Implemented in **DDSDataReaderStatusConditionHandler** (p. 1588).

9.242 DDSConditionSeq Struct Reference

Instantiates **FooSeq** (p. 1965) < **DDSCondition** (p. 1544) >

9.242.1 Detailed Description

Instantiates **FooSeq** (p. 1965) < **DDSCondition** (p. 1544) >

Instantiates:

<<***generic***>> *(*p. *433)* **FooSeq** (p. 1965)

See also

DDSWaitSet (p. 1897)

FooSeq (p. 1965)

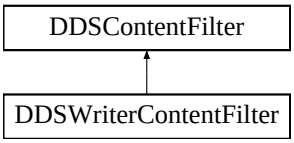
Examples

HelloWorld_subscriber.cxx.

9.243 DDSContentFilter Class Reference

<<***interface***>> *(*p. *433)* Interface to be used by a custom filter of a **DDSContentFilteredTopic** (p. 1551)

Inheritance diagram for DDSContentFilter:



Public Member Functions

- virtual **DDS_ReturnCode_t compile** (void **new_compile_data, const char *expression, const **DDS_String**↩ **Seq** &parameters, const **DDS_TypeCode** *type_code, const char *type_class_name, void *old_compile_↩ data)=0

  Compile an instance of the content filter according to the filter expression and parameters of the given data type.
- virtual **DDS_Boolean evaluate** (void *compile_data, const void *sample, const struct **DDS_FilterSampleInfo** *meta_data)=0

  Evaluate whether the sample is passing the filter or not according to the sample content.
- virtual void **finalize** (void *compile_data)=0

  A previously compiled instance of the content filter is no longer in use and resources can now be cleaned up.

9.243.1 Detailed Description

<<***interface***>> *(*p. *433)* Interface to be used by a custom filter of a **DDSContentFilteredTopic** (p. 1551)

Entity:

DDSContentFilteredTopic (p. 1551)

This interface can be implemented by an application-provided class and then registered with the **DDSDomain**↩ **Participant** (p. 1619) such that samples can be filtered for a **DDSContentFilteredTopic** (p. 1551) with the given filter name.

Note: the API for using a custom content filter is subject to change in a future release.

See also

DDSContentFilteredTopic (p. 1551)

DDSDomainParticipant::register_contentfilter (p. 1632)

9.243.2 Member Function Documentation

9.243.2.1 compile()

```
virtual  DDS_ReturnCode_t DDSContentFilter::compile (
             void ** new_compile_data,
             const char * expression,
             const  DDS_StringSeq & parameters,
             const  DDS_TypeCode * type_code,
             const char * type_class_name,
             void * old_compile_data )  [pure virtual]
```

Compile an instance of the content filter according to the filter expression and parameters of the given data type.

This method is called when an instance of the locally registered content filter is created or when the expression parameter for the locally registered content filter instance is changed.

An instance of the locally registered content filter is created every time a local **DDSContentFilteredTopic** (p. 1551) with the matching filter name is created, or when a **DDSDataReader** (p. 1556) with a matching filter name is discovered.

It is possible for multiple threads to be calling into this function at the same time. However, this function will never be called on a content filter that has been unregistered.

Parameters

new_compile_data	*<<**out**>> (p. 433)* User specified opaque pointer of this instance of the content filter. This value is then passed to the **DDSContentFilter::evaluate** (p. 1550) and **DDSContentFilter::finalize** (p. 1550) functions for this instance of the content filter. Can be set to NULL .
expression	*<<**in**>> (p. 433)* An ASCIIZ string with the filter expression. The memory used by this string is owned by RTI Connext and **must** not be freed. If you want to manipulate this string, you **must** first make a copy of it.
parameters	*<<**in**>> (p. 433)* A string sequence with the expression parameters the **DDSContentFilteredTopic** (p. 1551) was created with. The string sequence is **equal** (but **not** identical) to the string sequence passed to **DDSDomainParticipant::create_contentfilteredtopic** (p. 1653). Note that the sequence passed to the compile function is **owned** by RTI Connext and **must not** be referenced outside the compile function.
type_code	*<<**in**>> (p. 433)* A pointer to the type code for the related **DDSTopic** (p. 1885) of the **DDSContentFilteredTopic** (p. 1551). A type_code is a description of a type in terms of which types it contains (int32, string, etc.) and the corresponding member field names in the data type structure. The type code can be used to write custom content filters that can be used with any type.
type_class_name	*<<**in**>> (p. 433)* Fully qualified class name of the related **DDSTopic** (p. 1885).
old_compile_data	*<<**in**>> (p. 433)* The previous new_compile_data value from a previous call to this instance of a content filter. If the compile function is called more than once for an instance of a **DDSContentFilteredTopic** (p. 1551), e.g., if the expression parameters are changed, then the new_compile_data value returned by the previous invocation is passed in the old_compile_data parameter (which can be NULL). If this is a new instance of the filter, NULL is passed. This parameter is useful for freeing or reusing previously allocated resources.

Exceptions

One	of the **Standard Return Codes** (p. 532)

9.243.2.2 evaluate()

```
virtual  DDS_Boolean DDSContentFilter::evaluate (
             void * compile_data,
             const void * sample,
             const struct  DDS_FilterSampleInfo * meta_data )  [pure virtual]
```

Evaluate whether the sample is passing the filter or not according to the sample content.

This method is called when a sample for a locally created **DDSDataReader** (p. 1556) associated with the filter is received, or when a sample for a discovered **DDSDataReader** (p. 1556) assocated with the filter needs to be sent.

It is possible for multiple threads to be calling into this function at the same time. However, this function will never be called on a content filter that has been unregistered.

Parameters

compile_data	*<<**in**>> (p. 433)* The last return value of the **DDSContentFilter::compile** (p. 1548) function for this instance of the content filter. Can be NULL .
sample	*<<**in**>> (p. 433)* Pointer to a deserialized sample to be filtered
meta_data	*<<**in**>> (p. 433)* Pointer to meta data associated with the sample.

Returns

The function must return 0 if the sample should be filtered out, non zero otherwise

9.243.2.3 finalize()

```
virtual void DDSContentFilter::finalize (
             void * compile_data )  [pure virtual]
```

A previously compiled instance of the content filter is no longer in use and resources can now be cleaned up.

This method is called when an instance of the locally registered content filter is deleted.

An instance of the locally registered content filter is deleted every time a local **DDSContentFilteredTopic** (p. 1551) with the matching filter name is deleted, or when a **DDSDataReader** (p. 1556) with a matching filter name is removed due to discovery.

This method is also called on all instances of the discovered **DDSDataReader** (p. 1556) with a matching filter name if the filter is unregistered with **DDSDomainParticipant::unregister_contentfilter** (p. 1633).

It is possible for multiple threads to be calling into this function at the same time. However, this function will never be called on a content filter that has been unregistered.

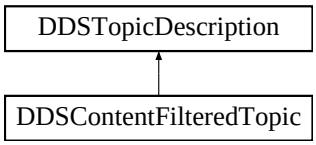
Parameters

compile_data	*<<**in**>> (p. 433)* The last return value of the **DDSContentFilter::compile** (p. 1548) function for this instance of the content filter. Can be NULL

9.244 DDSContentFilteredTopic Class Reference

*<<**interface**>> (p. 433)* Specialization of **DDSTopicDescription** (p. 1892) that allows for content-based subscriptions.

Inheritance diagram for DDSContentFilteredTopic:



Public Member Functions

- virtual const char * **get_filter_expression** ()=0

  Get the `filter_expression`.
- virtual **DDS_ReturnCode_t** **get_expression_parameters** (**DDS_StringSeq** &parameters)=0

  Get the `expression_parameters`.
- virtual **DDS_ReturnCode_t** **set_expression_parameters** (const **DDS_StringSeq** &parameters)=0

  Set the `expression_parameters`.
- virtual **DDS_ReturnCode_t** **set_expression** (const char *expression, const **DDS_StringSeq** &parameters)=0

  Set the `filter_expression` and `expression_parameters`.
- virtual **DDS_ReturnCode_t** **append_to_expression_parameter** (const **DDS_Long** index, const char *val)=0

  *<<**extension**>> (p. 432) Appends a string term to the specified parameter string.*
- virtual **DDS_ReturnCode_t** **remove_from_expression_parameter** (const **DDS_Long** index, const char *val)=0

  *<<**extension**>> (p. 432) Removes a string term from the specified parameter string.*
- virtual **DDSTopic** * **get_related_topic** ()=0

  Get the `related_topic`.

Static Public Member Functions

- static **DDSContentFilteredTopic** * **narrow** (**DDSTopicDescription** *topic_description)

  *Narrow the given **DDSTopicDescription** (p. 1892) pointer to a **DDSContentFilteredTopic** (p. 1551) pointer.*

9.244.1 Detailed Description

<<***interface***>> *(p. 433)* Specialization of **DDSTopicDescription** (p. 1892) that allows for content-based subscriptions.

It describes a more sophisticated subscription that indicates a **DDSDataReader** (p. 1556) does not want to necessarily see all values of each instance published under the **DDSTopic** (p. 1885). Rather, it wants to see only the values whose contents satisfy certain criteria. This class therefore can be used to request content-based subscriptions.

The selection of the content is done using the `filter_expression` with parameters `expression_parameters`.

- The `filter_expression` attribute is a string that specifies the criteria to select the data samples of interest. It is similar to the WHERE part of an SQL clause.

- The `expression_parameters` attribute is a sequence of strings that give values to the 'parameters' (i.e. "%n" tokens) in the `filter_expression`. The number of supplied parameters must fit with the requested values in the `filter_expression` (i.e. the number of n tokens).

Queries and Filters Syntax (p. 184) describes the syntax of `filter_expression` and `expression_parameters`.

9.244.2 Member Function Documentation

9.244.2.1 narrow()

```
static DDSContentFilteredTopic * DDSContentFilteredTopic::narrow (
            DDSTopicDescription * topic_description ) [static]
```

Narrow the given **DDSTopicDescription** (p. 1892) pointer to a **DDSContentFilteredTopic** (p. 1551) pointer.

Returns

DDSContentFilteredTopic (p. 1551) if this **DDSTopicDescription** (p. 1892) is a **DDSContentFilteredTopic** (p. 1551). Otherwise, return NULL.

9.244.2.2 get_filter_expression()

```
virtual const char * DDSContentFilteredTopic::get_filter_expression ( )  [pure virtual]
```

Get the `filter_expression`.

Return the `filter_expression` associated with the **DDSContentFilteredTopic** (p. 1551). `filter_expression` is either specified on the last successful call to **DDSContentFilteredTopic::set_expression** (p. 1554) or, if that method is never called, the expression specified when the **DDSContentFilteredTopic** (p. 1551) was created.

Returns

the `filter_expression`.

9.244.2.3 get_expression_parameters()

```
virtual DDS_ReturnCode_t DDSContentFilteredTopic::get_expression_parameters (
            DDS_StringSeq & parameters )  [pure virtual]
```

Get the `expression_parameters`.

Return the `expression_parameters` associated with the **DDSContentFilteredTopic** (p. 1551). `expression_parameters` is either specified on the last successful call to **DDSContentFilteredTopic::set_expression_parameters** (p. 1553), **DDSContentFilteredTopic::set_expression** (p. 1554) or, if that method is never called, the parameters specified when the **DDSContentFilteredTopic** (p. 1551) was created.

Parameters

parameters	<<***inout***>> *(p. 433)* the filter expression parameters. The memory for the strings in this sequence is managed according to the conventions described in **String Conventions** (p. 758). In particular, be careful to avoid a situation in which RTI Connext allocates a string on your behalf and you then reuse that string in such a way that RTI Connext believes it to have more memory allocated to it than it actually does.

Exceptions

One	of the **Standard Return Codes** (p. 532)

See also

DDSDomainParticipant::create_contentfilteredtopic (p. 1653)

DDSContentFilteredTopic::set_expression_parameters (p. 1553)

9.244.2.4 set_expression_parameters()

```
virtual DDS_ReturnCode_t DDSContentFilteredTopic::set_expression_parameters (
            const DDS_StringSeq & parameters ) [pure virtual]
```

Set the expression_parameters.

Change the expression_parameters associated with the **DDSContentFilteredTopic** (p. 1551).

Parameters

parameters	<<***in***>> *(p. 433)* the filter expression parameters Length of sequence cannot be greater than 100.

Exceptions

One	of the **Standard Return Codes** (p. 532)

9.244.2.5 set_expression()

```
virtual DDS_ReturnCode_t DDSContentFilteredTopic::set_expression (
            const char * expression,
            const DDS_StringSeq & parameters ) [pure virtual]
```

Set the filter_expression and expression_parameters.

Changes the filter_expression and expression_parameters associated with the **DDSContentFilteredTopic** (p. 1551).

Parameters

expression	<<***in***>> *(p. 433)* the filter expression.
parameters	<<***in***>> *(p. 433)* the filter expression parameters Length of sequence cannot be greater than 100.

Returns

One of the **Standard Return Codes** (p. 532)

9.244.2.6 append_to_expression_parameter()

```
virtual DDS_ReturnCode_t DDSContentFilteredTopic::append_to_expression_parameter (
            const DDS_Long index,
            const char * val ) [pure virtual]
```

*<<**extension**>> (p. 432)* Appends a string term to the specified parameter string.

Appends the input string to the end of the specified parameter string, separated by a comma. If the original parameter string is enclosed in quotation marks ("), the resultant string will also be enclosed in quotation marks.

This method can be used in expression parameters associated with MATCH operators in order to add a pattern to the match pattern list. For example, if the filter expression parameter value is:

'IBM'

Then append_to_expression_parameter(0, "MSFT") would generate the new value:

'IBM,MSFT'

Parameters

index	*<<**in**>> (p. 433)* The index of the parameter string to be modified. The first index is index 0. When using the **DDS_STRINGMATCHFILTER_NAME** (p. 61) filter, `index` *must* be 0.
val	*<<**in**>> (p. 433)* The string term to be appended to the parameter string.

Exceptions

One	of the **Standard Return Codes** (p. 532)

9.244.2.7 remove_from_expression_parameter()

```
virtual  DDS_ReturnCode_t DDSContentFilteredTopic::remove_from_expression_parameter (
             const  DDS_Long index,
             const char * val )  [pure virtual]
```

*<<**extension**>> (p. 432)* Removes a string term from the specified parameter string.

Removes the input string from the specified parameter string. To be found and removed, the input string must exist as a complete term, bounded by comma separators or the strong boundary. If the original parameter string is enclosed in quotation marks ("), the resultant string will also be enclosed in quotation marks. If the removed term was the last entry in the string, the result will be a string of empty quotation marks.

This method can be used in expression parameters associated with MATCH operators in order to remove a pattern from the match pattern list. For example, if the filter expression paremeter value is:

'IBM,MSFT'

Then remove_from_expression_parameter(0, "IBM") would generate the expression:

'MSFT'

Parameters

index	<<***in***>> *(*p. *433)* The index of the parameter string to be modified. The first index is index 0. When using the **DDS_STRINGMATCHFILTER_NAME** (p. 61) filter, `index` *must* be 0.
val	<<***in***>> *(*p. *433)* The string term to be removed from the parameter string.

Exceptions

One	of the **Standard Return Codes** (p. 532)

9.244.2.8 get_related_topic()

```
virtual DDSTopic * DDSContentFilteredTopic::get_related_topic ( ) [pure virtual]
```

Get the `related_topic`.

Return the **DDSTopic** (p. 1885) specified when the **DDSContentFilteredTopic** (p. 1551) was created.

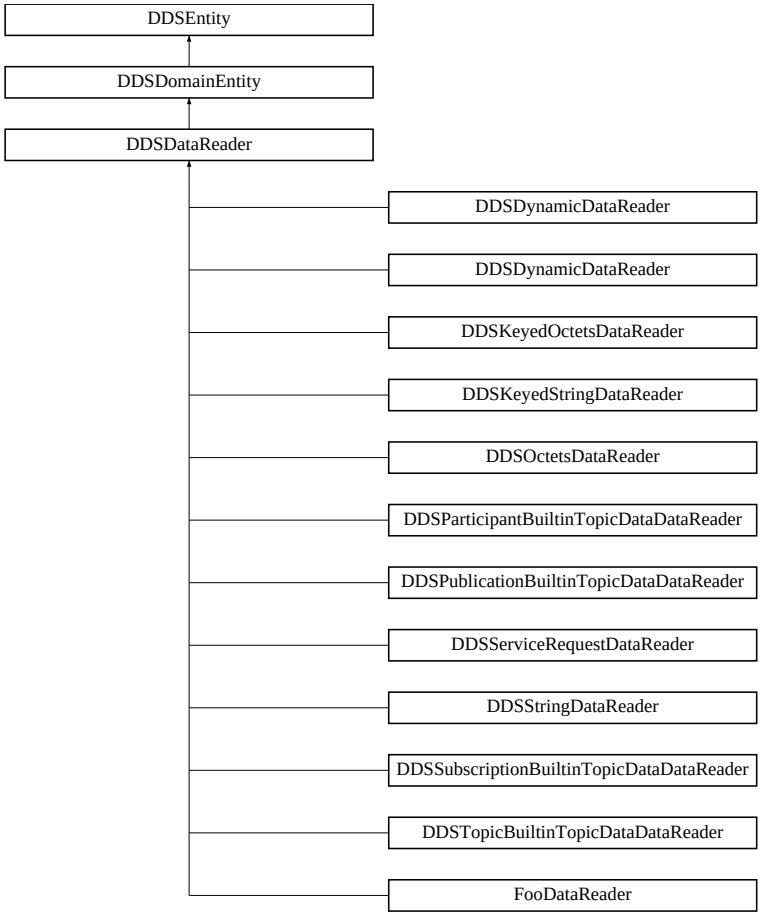
Returns

The **DDSTopic** (p. 1885) assocated with the **DDSContentFilteredTopic** (p. 1551).

9.245 DDSDataReader Class Reference

<<***interface***>> *(*p. *433)* Allows the application to: (1) declare the data it wishes to receive (i.e. make a subscription) and (2) access the data received by the attached **DDSSubscriber** (p. 1860).

Inheritance diagram for DDSDataReader:



Public Member Functions

- virtual **DDSReadCondition** * **create_readcondition** (**DDS_SampleStateMask** sample_states, **DDS_View↩ StateMask** view_states, **DDS_InstanceStateMask** instance_states)

  *Creates a **DDSReadCondition** (p. 1842).*
- virtual **DDSReadCondition** * **create_readcondition_w_params** (**DDS_ReadConditionParams** &params)

  *<<**extension**>> (p. 432) Creates a **DDSReadCondition** (p. 1842) with parameters.*
- virtual **DDSQueryCondition** * **create_querycondition** (**DDS_SampleStateMask** sample_states, **DDS_↩ ViewStateMask** view_states, **DDS_InstanceStateMask** instance_states, const char *query_expression, const struct **DDS_StringSeq** &query_parameters)

  *Creates a **DDSQueryCondition** (p. 1841).*
- virtual **DDSQueryCondition** * **create_querycondition_w_params** (**DDS_QueryConditionParams** &params)

  *<<**extension**>> (p. 432) Creates a **DDSQueryCondition** (p. 1841) with parameters.*
- virtual **DDS_ReturnCode_t** **delete_readcondition** (**DDSReadCondition** *condition)

  *Deletes a **DDSReadCondition** (p. 1842) or **DDSQueryCondition** (p. 1841) attached to the **DDSDataReader** (p. 1556).*
- virtual **DDS_ReturnCode_t** **delete_contained_entities** ()

  *Deletes all the entities that were created by means of the "create" operations on the **DDSDataReader** (p. 1556).*
- virtual **DDS_ReturnCode_t** **wait_for_historical_data** (const **DDS_Duration_t** &max_wait)

  *Waits until all "historical" data is received for **DDSDataReader** (p. 1556) entities that have a non-VOLATILE Durability Qos kind.*

- virtual **DDS_ReturnCode_t acknowledge_sample** (const **DDS_SampleInfo** &sample_info)

  <<***extension***>> (p. *432) Acknowledges a single sample explicitly.*
- virtual **DDS_ReturnCode_t acknowledge_all** ()

  <<***extension***>> (p. *432) Acknowledges all previously accessed samples.*
- virtual **DDS_ReturnCode_t acknowledge_sample** (const **DDS_SampleInfo** &sample_info, const **DDS_↩ AckResponseData_t** &response_data)

  <<***extension***>> (p. *432) Acknowledges a single sample explicitly.*
- virtual **DDS_ReturnCode_t acknowledge_all** (const **DDS_AckResponseData_t** &response_data)

  <<***extension***>> (p. *432) Acknowledges all previously accessed samples.*
- virtual **DDS_ReturnCode_t get_matched_publications** (**DDS_InstanceHandleSeq** &publication_handles)

  *Retrieves the list of publications currently "associated" with this **DDSDataReader** (p. 1556).*
- virtual **DDS_ReturnCode_t get_matched_publication_data** (**DDS_PublicationBuiltinTopicData** &publication_data, const **DDS_InstanceHandle_t** &publication_handle)

  *Retrieves the information on a publication that is currently "associated" with the **DDSDataReader** (p. 1556).*
- virtual **DDS_ReturnCode_t is_matched_publication_alive** (**DDS_Boolean** &is_alive, const **DDS_Instance↩ Handle_t** &publication_handle)

  Check if a publication currently matched with a DataReader is alive.
- virtual **DDS_ReturnCode_t get_matched_publication_participant_data** (**DDS_ParticipantBuiltinTopicData** &participant_data, const **DDS_InstanceHandle_t** &publication_handle)

  <<***extension***>> (p. *432) Retrieves the information on the discovered **DDSDomainParticipant** (p. 1619) associated with the publication that is currently matching with the **DDSDataReader** (p. 1556).*
- virtual **DDSTopicDescription** * **get_topicdescription** ()

  *Returns the **DDSTopicDescription** (p. 1892) associated with the **DDSDataReader** (p. 1556).*
- virtual **DDSSubscriber** * **get_subscriber** ()

  *Returns the **DDSSubscriber** (p. 1860) to which the **DDSDataReader** (p. 1556) belongs.*
- virtual **DDS_ReturnCode_t get_sample_rejected_status** (**DDS_SampleRejectedStatus** &status)

  *Accesses the **DDS_SAMPLE_REJECTED_STATUS** (p. 541) communication status.*
- virtual **DDS_ReturnCode_t get_liveliness_changed_status** (**DDS_LivelinessChangedStatus** &status)

  *Accesses the **DDS_LIVELINESS_CHANGED_STATUS** (p. 542) communication status.*
- virtual **DDS_ReturnCode_t get_requested_deadline_missed_status** (**DDS_RequestedDeadlineMissed↩ Status** &status)

  *Accesses the **DDS_REQUESTED_DEADLINE_MISSED_STATUS** (p. 540) communication status.*
- virtual **DDS_ReturnCode_t get_requested_incompatible_qos_status** (**DDS_RequestedIncompatible↩ QosStatus** &status)

  *Accesses the **DDS_REQUESTED_INCOMPATIBLE_QOS_STATUS** (p. 540) communication status.*
- virtual **DDS_ReturnCode_t get_sample_lost_status** (**DDS_SampleLostStatus** &status)

  *Accesses the **DDS_SAMPLE_LOST_STATUS** (p. 541) communication status.*
- virtual **DDS_ReturnCode_t get_subscription_matched_status** (**DDS_SubscriptionMatchedStatus** &status)

  *Accesses the **DDS_SUBSCRIPTION_MATCHED_STATUS** (p. 543) communication status.*
- virtual **DDS_ReturnCode_t get_datareader_cache_status** (**DDS_DataReaderCacheStatus** &status)

  <<***extension***>> (p. *432) Gets the datareader cache status for this reader.*
- virtual **DDS_ReturnCode_t get_datareader_protocol_status** (**DDS_DataReaderProtocolStatus** &status)

  <<***extension***>> (p. *432) Gets the datareader protocol status for this reader.*
- virtual **DDS_ReturnCode_t get_matched_publication_datareader_protocol_status** (**DDS_DataReader↩ ProtocolStatus** &status, const **DDS_InstanceHandle_t** &publication_handle)

  <<***extension***>> (p. *432) Gets the datareader protocol status for this reader, per matched publication identified by the publication_handle.*
- virtual **DDS_ReturnCode_t set_qos** (const **DDS_DataReaderQos** &qos)

Sets the reader QoS.

- virtual **DDS_ReturnCode_t get_qos** (**DDS_DataReaderQos** &qos)

  Gets the reader QoS.

- virtual **DDS_ReturnCode_t set_qos_with_profile** (const char *library_name, const char *profile_name)

  *<<**extension**>> (p. 432) Changes the QoS of this reader using the input XML QoS profile.*

- virtual **DDS_ReturnCode_t set_property** (const char *property_name, const char *value, bool propagate)

  Set the value for a property that applies to a DataReader.

- virtual **DDS_ReturnCode_t set_listener** (**DDSDataReaderListener** *l, **DDS_StatusMask** mask= **DDS_STATUS_MASK_ALL**)

  Sets the reader listener.

- virtual **DDSDataReaderListener** * **get_listener** ()

  Gets the reader listener.

- virtual **DDSTopicQuery** * **create_topic_query** (const **DDS_TopicQuerySelection** &selection)

  *Creates a **DDSTopicQuery** (p. 1895).*

- virtual **DDS_ReturnCode_t delete_topic_query** (**DDSTopicQuery** *query)

  *Deletes a **DDSTopicQuery** (p. 1895).*

- virtual **DDSTopicQuery** * **lookup_topic_query** (const **DDS_GUID_t** &guid)

  *Retrieves an existing **DDSTopicQuery** (p. 1895).*

- virtual **DDS_ReturnCode_t take_discovery_snapshot** ()

  Take a snapshot of the compatible and incompatible remote writers matched by a local reader.

- virtual **DDS_ReturnCode_t take_discovery_snapshot** (const char *file_name)

  Take a snapshot of the compatible and incompatible remote writers matched by a local reader.

- virtual **DDS_ReturnCode_t enable** ()

  *Enables the **DDSEntity** (p. 1731).*

- virtual **DDSStatusCondition** * **get_statuscondition** ()

  *Allows access to the **DDSStatusCondition** (p. 1846) associated with the **DDSEntity** (p. 1731).*

- virtual **DDS_StatusMask get_status_changes** ()

  *Retrieves the list of communication statuses in the **DDSEntity** (p. 1731) that are triggered.*

- virtual **DDS_InstanceHandle_t get_instance_handle** ()

  *Allows access to the **DDS_InstanceHandle_t** (p. 76) associated with the **DDSEntity** (p. 1731).*

9.245.1 Detailed Description

*<<**interface**>> (p. 433)* Allows the application to: (1) declare the data it wishes to receive (i.e. make a subscription) and (2) access the data received by the attached **DDSSubscriber** (p. 1860).

QoS:

DDS_DataReaderQos (p. 858)

Status:

DDS_DATA_AVAILABLE_STATUS (p. 541);
DDS_LIVELINESS_CHANGED_STATUS (p. 542), **DDS_LivelinessChangedStatus** (p. 1176);
DDS_REQUESTED_DEADLINE_MISSED_STATUS (p. 540), **DDS_RequestedDeadlineMissedStatus** (p. 1306);
DDS_REQUESTED_INCOMPATIBLE_QOS_STATUS (p. 540), **DDS_RequestedIncompatibleQosStatus** (p. 1307);
DDS_SAMPLE_LOST_STATUS (p. 541), **DDS_SampleLostStatus** (p. 1355);
DDS_SAMPLE_REJECTED_STATUS (p. 541), **DDS_SampleRejectedStatus** (p. 1356);
DDS_SUBSCRIPTION_MATCHED_STATUS (p. 543), **DDS_SubscriptionMatchedStatus** (p. 1379);

Listener:

DDSDataReaderListener (p. 1583)

A **DDSDataReader** (p. 1556) refers to exactly one **DDSTopicDescription** (p. 1892) (either a **DDSTopic** (p. 1885), a **DDSContentFilteredTopic** (p. 1551) or a **DDSMultiTopic** (p. 1797)) that identifies the data to be read.

The subscription has a unique resulting type. The data-reader may give access to several instances of the resulting type, which can be distinguished from each other by their `key`.

DDSDataReader (p. 1556) is an abstract class. It must be specialised for each particular application data-type (see **USER_DATA** (p. 653)). The additional methods or functions that must be defined in the auto-generated class for a hypothetical application type **`Foo`** (p. 1916) are specified in the generic type **FooDataReader** (p. 1917).

The following operations may be called even if the **DDSDataReader** (p. 1556) is not enabled. Other operations will fail with the value **DDS_RETCODE_NOT_ENABLED** (p. 533) if called on a disabled **DDSDataReader** (p. 1556):

- The base-class operations **DDSDataReader::set_qos** (p. 1574), **DDSDataReader::get_qos** (p. 1575), **DDSDataReader::set_listener** (p. 1577), **DDSDataReader::get_listener** (p. 1577), **DDSEntity::enable** (p. 1734), **DDSEntity::get_statuscondition** (p. 1735), **DDSEntity::get_status_changes** (p. 1735),
- **DDSDataReader::get_liveliness_changed_status** (p. 1570), **DDSDataReader::get_requested_deadline_missed_status** (p. 1571), **DDSDataReader::get_requested_incompatible_qos_status** (p. 1571), **DDSDataReader::get_sample_lost_status** (p. 1572), **DDSDataReader::get_sample_rejected_status** (p. 1570), **DDSDataReader::get_subscription_matched_status** (p. 1572)

All sample-accessing operations, namely: **FooDataReader::read** (p. 1919), **FooDataReader::take** (p. 1920), **FooDataReader::read_w_condition** (p. 1925), and **FooDataReader::take_w_condition** (p. 1926) may fail with the error **DDS_RETCODE_PRECONDITION_NOT_MET** (p. 532) as described in **DDSSubscriber::begin_access** (p. 1873).

See also

Operations Allowed in Listener Callbacks (p. 1796)

Access to data samples (p. 129)

Examples

HelloWorldSupport.cxx, and **HelloWorld_subscriber.cxx**.

9.245.2 Member Function Documentation

9.245.2.1 create_readcondition()

```
virtual  DDSReadCondition * DDSDataReader::create_readcondition (
              DDS_SampleStateMask sample_states,
              DDS_ViewStateMask view_states,
              DDS_InstanceStateMask instance_states )  [inline], [virtual]
```

Creates a **DDSReadCondition** (p. 1842).

The returned **DDSReadCondition** (p. 1842) will be attached and belong to the **DDSDataReader** (p. 1556).

Parameters

sample_states	<<***in***>> *(p. 433)* sample state of the data samples that are of interest
view_states	<<***in***>> *(p. 433)* view state of the data samples that are of interest
instance_states	<<***in***>> *(p. 433)* instance state of the data samples that are of interest

Returns

return **DDSReadCondition** (p. 1842) created. Returns NULL in case of failure.

9.245.2.2 create_readcondition_w_params()

```
virtual  DDSReadCondition * DDSDataReader::create_readcondition_w_params (
              DDS_ReadConditionParams & params )  [inline], [virtual]
```

<<***extension***>> *(p. 432)* Creates a **DDSReadCondition** (p. 1842) with parameters.

The returned **DDSReadCondition** (p. 1842) will be attached and belong to the **DDSDataReader** (p. 1556).

Parameters

params	<<***in***>> *(p. 433)* creation parameters.

Returns

return **DDSReadCondition** (p. 1842) created. Returns NULL in case of failure.

9.245.2.3 create_querycondition()

```
virtual DDSQueryCondition * DDSDataReader::create_querycondition (
            DDS_SampleStateMask sample_states,
            DDS_ViewStateMask view_states,
            DDS_InstanceStateMask instance_states,
            const char * query_expression,
            const struct DDS_StringSeq & query_parameters ) [inline], [virtual]
```

Creates a **DDSQueryCondition** (p. 1841).

The returned **DDSQueryCondition** (p. 1841) will be attached and belong to the **DDSDataReader** (p. 1556).

Queries and Filters Syntax (p. 184) describes the syntax of `query_expression` and `query_parameters`.

Parameters

sample_states	<<***in***>> *(p. 433)* sample state of the data samples that are of interest
view_states	<<***in***>> *(p. 433)* view state of the data samples that are of interest
instance_states	<<***in***>> *(p. 433)* instance state of the data samples that are of interest
query_expression	<<***in***>> *(p. 433)* Expression for the query. Cannot be NULL.
query_parameters	<<***in***>> *(p. 433)* Parameters for the query expression.

Returns

return **DDSQueryCondition** (p. 1841) created. Returns NULL in case of failure.

9.245.2.4 create_querycondition_w_params()

```
virtual DDSQueryCondition * DDSDataReader::create_querycondition_w_params (
            DDS_QueryConditionParams & params ) [inline], [virtual]
```

<<***extension***>> *(p. 432)* Creates a **DDSQueryCondition** (p. 1841) with parameters.

The returned **DDSQueryCondition** (p. 1841) will be attached and belong to the **DDSDataReader** (p. 1556).

Parameters

params	<<***in***>> *(p. 433)* creation parameters.

Returns

return **DDSQueryCondition** (p. 1841) created. Returns NULL in case of failure.

9.245.2.5 delete_readcondition()

```
virtual DDS_ReturnCode_t DDSDataReader::delete_readcondition (
            DDSReadCondition * condition ) [inline], [virtual]
```

Deletes a **DDSReadCondition** (p. 1842) or **DDSQueryCondition** (p. 1841) attached to the **DDSDataReader** (p. 1556).

Since **DDSQueryCondition** (p. 1841) specializes **DDSReadCondition** (p. 1842), it can also be used to delete a **DDSQueryCondition** (p. 1841).

Precondition

The **DDSReadCondition** (p. 1842) must be attached to the **DDSDataReader** (p. 1556), or the operation will fail with the error **DDS_RETCODE_PRECONDITION_NOT_MET** (p. 532).

MT Safety:

UNSAFE. It is not safe to delete an entity while another thread may be simultaneously calling an API that uses the entity.

Parameters

condition	<<***in***>> *(p. 433)* Condition to be deleted.

Exceptions

One	of the **Standard Return Codes** (p. 532), or **DDS_RETCODE_PRECONDITION_NOT_MET** (p. 532)

9.245.2.6 delete_contained_entities()

```
virtual DDS_ReturnCode_t DDSDataReader::delete_contained_entities ( ) [inline], [virtual]
```

Deletes all the entities that were created by means of the "create" operations on the **DDSDataReader** (p. 1556).

Deletes all contained **DDSReadCondition** (p. 1842), **DDSQueryCondition** (p. 1841), and **DDSTopicQuery** (p. 1895) objects.

The operation will fail with **DDS_RETCODE_PRECONDITION_NOT_MET** (p. 532) if the any of the contained entities is in a state where it cannot be deleted.

Once **DDSDataReader::delete_contained_entities** (p. 1563) completes successfully, the application may delete the **DDSDataReader** (p. 1556).

MT Safety:

UNSAFE. It is not safe to delete an entity while another thread may be simultaneously calling an API that uses the entity.

Exceptions

One	of the **Standard Return Codes** (p. 532), or **DDS_RETCODE_PRECONDITION_NOT_MET** (p. 532)

9.245.2.7 wait_for_historical_data()

```
virtual DDS_ReturnCode_t DDSDataReader::wait_for_historical_data (
            const DDS_Duration_t & max_wait ) [inline], [virtual]
```

Waits until all "historical" data is received for **DDSDataReader** (p. 1556) entities that have a non-VOLATILE Durability Qos kind.

This operation is intended only for **DDSDataReader** (p. 1556) entities that have a non-VOLATILE Durability QoS kind.

As soon as an application enables a non-VOLATILE **DDSDataReader** (p. 1556), it will start receiving both "historical" data (i.e., the data that was written prior to the time the **DDSDataReader** (p. 1556) joined the domain) as well as any new data written by the **DDSDataWriter** (p. 1589) entities. There are situations where the application logic may require the application to wait until all "historical" data is received. This is the purpose of the **DDSDataReader::wait_for_historical_data** (p. 1564) operations.

DDSDataReader::wait_for_historical_data() (p. 1564) blocks the calling thread until either all "historical" data is received, or until the duration specified by the max_wait parameter elapses, whichever happens first. It will return immediately if no DataWriters have been discovered at the time the operation is called; therefore it is advisable to make sure at least one **DDSDataWriter** (p. 1589) has been discovered before calling this operation. (One way to do this is by using **DDSDataReader::get_subscription_matched_status** (p. 1572).)

A successful completion indicates that all the "historical" data was "received"; timing out indicates that max_wait elapsed before all the data was received.

Parameters

max_wait	<<***in***>> *(p. 433)* Timeout value.

Exceptions

One	of the **Standard Return Codes** (p. 532), **DDS_RETCODE_TIMEOUT** (p. 533) or **DDS_RETCODE_NOT_ENABLED** (p. 533).

9.245.2.8 acknowledge_sample() [1/2]

```
virtual DDS_ReturnCode_t DDSDataReader::acknowledge_sample (
            const DDS_SampleInfo & sample_info ) [inline], [virtual]
```

<<***extension***>> *(p. 432)* Acknowledges a single sample explicitly.

Applicable only when **DDS_ReliabilityQosPolicy::acknowledgment_kind** (p. 1298) = **DDS_APPLICATION_↩ EXPLICIT_ACKNOWLEDGMENT_MODE** (p. 634)

A call to this method does not necessarily trigger the sending of an AppAck RTPS message from the DataReader to the DataWriter. How and when AppAck messages are sent can be configured using the QoS values **DDS_Rtps↩ ReliableReaderProtocol_t::samples_per_app_ack** (p. 1319) and **DDS_RtpsReliableReaderProtocol_t::app_ack↩ _period** (p. 1318).

Parameters

sample_info	<<***in***>> *(p. 433)* **DDS_SampleInfo** (p. 1344) identifying the sample being acknowledged.

Exceptions

One	of the **Standard Return Codes** (p. 532)

9.245.2.9 acknowledge_all() [1/2]

```
virtual DDS_ReturnCode_t DDSDataReader::acknowledge_all ( ) [inline], [virtual]
```

<<***extension***>> *(p. 432)* Acknowledges all previously accessed samples.

Applicable only when **DDS_ReliabilityQosPolicy::acknowledgment_kind** (p. 1298) = **DDS_APPLICATION_↩ EXPLICIT_ACKNOWLEDGMENT_MODE** (p. 634)

A call to this method does not necessarily trigger the sending of an AppAck RTPS message from the DataReader to the DataWriter. How and when AppAck messages are sent can be configured using the QoS values **DDS_Rtps↩ ReliableReaderProtocol_t::samples_per_app_ack** (p. 1319) and **DDS_RtpsReliableReaderProtocol_t::app_ack↩ _period** (p. 1318).

Exceptions

One	of the **Standard Return Codes** (p. 532)

9.245.2.10 acknowledge_sample() [2/2]

```
virtual DDS_ReturnCode_t DDSDataReader::acknowledge_sample (
            const DDS_SampleInfo & sample_info,
            const DDS_AckResponseData_t & response_data ) [inline], [virtual]
```

<<***extension***>> *(p. 432)* Acknowledges a single sample explicitly.

Applicable only when **DDS_ReliabilityQosPolicy::acknowledgment_kind** (p. 1298) = **DDS_APPLICATION_EXPLICIT_ACKNOWLEDGMENT_MODE** (p. 634)

A call to this method does not necessarily trigger the sending of an AppAck RTPS message from the DataReader to the DataWriter. How and when AppAck messages are sent can be configured using the QoS values **DDS_RtpsReliableReaderProtocol_t::samples_per_app_ack** (p. 1319) and **DDS_RtpsReliableReaderProtocol_t::app_ack_period** (p. 1318).

The maximum length of the response is configured using **DDS_DataReaderResourceLimitsQosPolicy::max_app_ack_response_length** (p. 881)

Parameters

sample_info	<<***in***>> *(p. 433)* **DDS_SampleInfo** (p. 1344) identifying the sample being acknowledged.
response_data	<<***in***>> *(p. 433)* Response data sent to **DDSDataWriter** (p. 1589) upon acknowledgment (via **DDSDataWriterListener::on_application_acknowledgment** (p. 1617))

Exceptions

One	of the **Standard Return Codes** (p. 532)

9.245.2.11 acknowledge_all() [2/2]

```
virtual DDS_ReturnCode_t DDSDataReader::acknowledge_all (
            const DDS_AckResponseData_t & response_data ) [inline], [virtual]
```

<<***extension***>> *(p. 432)* Acknowledges all previously accessed samples.

Applicable only when **DDS_ReliabilityQosPolicy::acknowledgment_kind** (p. 1298) = **DDS_APPLICATION_EXPLICIT_ACKNOWLEDGMENT_MODE** (p. 634)

A call to this method does not necessarily trigger the sending of an AppAck RTPS message from the DataReader to the DataWriter. How and when AppAck messages are sent can be configured using the QoS values **DDS_RtpsReliableReaderProtocol_t::samples_per_app_ack** (p. 1319) and **DDS_RtpsReliableReaderProtocol_t::app_ack_period** (p. 1318).

The maximum length of the response is configured using **DDS_DataReaderResourceLimitsQosPolicy::max_app_ack_response_length** (p. 881).

Parameters

response_data	<<***in***>> *(p. 433)* Response data sent to **DDSDataWriter** (p. 1589) upon acknowledgment

Exceptions

One	of the **Standard Return Codes** (p. 532)

9.245.2.12 get_matched_publications()

```
virtual  DDS_ReturnCode_t DDSDataReader::get_matched_publications (
              DDS_InstanceHandleSeq & publication_handles )  [inline], [virtual]
```

Retrieves the list of publications currently "associated" with this **DDSDataReader** (p. 1556).

A publication is considered to be matching if all of the following criteria are true:

- The publication is within the same domain as this subscription.
- The publication has a matching **DDSTopic** (p. 1885).
- The publication has compatible QoS.
- If the applications are using partitions, the publication shares a common partition with this subscription.
- The **DDSDomainParticipant** (p. 1619) has not indicated that the publication's **DDSDomainParticipant** (p. 1619) should be "ignored" by means of the **DDSDomainParticipant::ignore_publication** (p. 1664) API.
- If the subscription is using a **DDSContentFilteredTopic** (p. 1551) and the publication is using the **DDS_Multi↩ ChannelQosPolicy** (p. 1214), there is an intersection between at least one of the associated filters.
- If the endpoints need to exchange key material to communicate (i.e., they are securing their communications), the reader has completed the key exchange with the writer.

The handles returned in the `publication_handles` list are the ones that are used by the RTI Connext implementation to locally identify the corresponding matched **DDSDataWriter** (p. 1589) entities. These handles match the ones that appear in the `instance_handle` field of the **DDS_SampleInfo** (p. 1344) when reading the **DDS_PUBLICATION↩ _TOPIC_NAME** (p. 494) builtin topic.

This API may return the publication handles of publications that are not alive. **DDSDataReader::is_matched_↩ publication_alive** (p. 1568) can be used to check the liveliness of the remote publication.

Parameters

publication_handles	*<<**inout**>> (p. 433).* The sequence will be grown if the sequence has ownership and the system has the corresponding resources. Use a sequence without ownership to avoid dynamic memory allocation. If the sequence is too small to store all of the matches and the system cannot resize the sequence, this method will fail with **DDS_RETCODE_OUT_OF_RESOURCES** (p. 533). The maximum number of matches possible is configured with **DDS_DomainParticipantResourceLimitsQosPolicy** (p. 977). You can use a zero-maximum sequence without ownership to quickly check whether there are any matches without allocating any memory.

Exceptions

One	of the **Standard Return Codes** (p. 532), or **DDS_RETCODE_OUT_OF_RESOURCES** (p. 533) if the sequence is too small and the system cannot resize it, or **DDS_RETCODE_NOT_ENABLED** (p. 533)

9.245.2.13 get_matched_publication_data()

```
virtual  DDS_ReturnCode_t DDSDataReader::get_matched_publication_data (
              DDS_PublicationBuiltinTopicData & publication_data,
            const  DDS_InstanceHandle_t & publication_handle )  [inline], [virtual]
```

Retrieves the information on a publication that is currently "associated" with the **DDSDataReader** (p. 1556).

The `publication_handle` must correspond to a publication currently associated with the **DDSDataReader** (p. 1556). Otherwise, the operation will fail with **DDS_RETCODE_BAD_PARAMETER** (p. 532). Use the operation **DDSDataReader::get_matched_publications** (p. 1567) to find the publications that are currently matched with the **DDSDataReader** (p. 1556).

Note: This operation does not retrieve the **DDS_PublicationBuiltinTopicData::type_code** (p. 1269). This information is available through **DDSDataReaderListener::on_data_available()** (p. 1585) (if a reader listener is installed on the **DDSPublicationBuiltinTopicDataDataReader** (p. 1817)).

Parameters

publication_data	*<<**inout**>> (p. 433).* The information to be filled in on the associated publication. Cannot be NULL.
publication_handle	*<<**in**>> (p. 433).* Handle to a specific publication associated with the **DDSDataWriter** (p. 1589). Must correspond to a publication currently associated with the **DDSDataReader** (p. 1556).

Exceptions

One	of the **Standard Return Codes** (p. 532) or **DDS_RETCODE_NOT_ENABLED** (p. 533)

9.245.2.14 is_matched_publication_alive()

```
virtual  DDS_ReturnCode_t DDSDataReader::is_matched_publication_alive (
              DDS_Boolean & is_alive,
            const  DDS_InstanceHandle_t & publication_handle )  [inline], [virtual]
```

Check if a publication currently matched with a DataReader is alive.

This API is used for querying the endpoint liveliness of a matched publication. A matched publication will be marked as not alive if the liveliness that it committed to through its **LIVELINESS** (p. 607) QoS policy was not respected. Note that if the participant associated with the matched publication loses liveliness, the **DDS_InstanceHandle_t** (p. 76) will become invalid and this function will fail with **DDS_RETCODE_BAD_PARAMETER** (p. 532).

Parameters

publication_handle	*<<**in**>> (p. 433)* The **DDS_InstanceHandle_t** (p. 76) of the matched publication. See **DDSDataReader::get_matched_publications** (p. 1567) for a description of what is considered a matched publication.
is_alive	*<<**out**>> (p. 433)* Indicates whether or not the matched publication is alive.

Exceptions

One	of the **Standard Return Codes** (p. 532)

9.245.2.15 get_matched_publication_participant_data()

```
virtual DDS_ReturnCode_t DDSDataReader::get_matched_publication_participant_data (
            DDS_ParticipantBuiltinTopicData & participant_data,
          const DDS_InstanceHandle_t & publication_handle ) [inline], [virtual]
```

*<<**extension**>> (p. 432)* Retrieves the information on the discovered **DDSDomainParticipant** (p. 1619) associated with the publication that is currently matching with the **DDSDataReader** (p. 1556).

The `publication_handle` must correspond to a publication currently associated with the **DDSDataReader** (p. 1556). Otherwise, the operation will fail with **DDS_RETCODE_BAD_PARAMETER** (p. 532). The operation may also fail with **DDS_RETCODE_PRECONDITION_NOT_MET** (p. 532) if the publication corresponds to the same **DDSDomainParticipant** (p. 1619) that the DataReader belongs to. Use the operation **DDSDataReader::get_↩ matched_publications** (p. 1567) to find the publications that are currently matched with the **DDSDataReader** (p. 1556).

Parameters

participant_data	*<<**inout**>> (p. 433).* The information to be filled in on the associated participant. Cannot be NULL.
publication_handle	*<<**in**>> (p. 433).* Handle to a specific publication associated with a **DDSDataWriter** (p. 1589). Must correspond to a publication currently associated with the **DDSDataReader** (p. 1556).

Exceptions

One	of the **Standard Return Codes** (p. 532) or **DDS_RETCODE_NOT_ENABLED** (p. 533)

9.245.2.16 get_topicdescription()

```
virtual DDSTopicDescription * DDSDataReader::get_topicdescription ( ) [inline], [virtual]
```

Returns the **DDSTopicDescription** (p. 1892) associated with the **DDSDataReader** (p. 1556).

Returns that same **DDSTopicDescription** (p. 1892) that was used to create the **DDSDataReader** (p. 1556).

Returns

DDSTopicDescription (p. 1892) associated with the **DDSDataReader** (p. 1556).

9.245.2.17 get_subscriber()

```
virtual DDSSubscriber * DDSDataReader::get_subscriber ( ) [inline], [virtual]
```

Returns the **DDSSubscriber** (p. 1860) to which the **DDSDataReader** (p. 1556) belongs.

Returns

DDSSubscriber (p. 1860) to which the **DDSDataReader** (p. 1556) belongs.

9.245.2.18 get_sample_rejected_status()

```
virtual DDS_ReturnCode_t DDSDataReader::get_sample_rejected_status (
            DDS_SampleRejectedStatus & status ) [inline], [virtual]
```

Accesses the **DDS_SAMPLE_REJECTED_STATUS** (p. 541) communication status.

This also resets the status so that it is no longer considered changed.

Parameters

status	<<***inout***>> *(p. 433)* **DDS_SampleRejectedStatus** (p. 1356) to be filled in.

Exceptions

One	of the **Standard Return Codes** (p. 532)

9.245.2.19 get_liveliness_changed_status()

```
virtual DDS_ReturnCode_t DDSDataReader::get_liveliness_changed_status (
            DDS_LivelinessChangedStatus & status ) [inline], [virtual]
```

Accesses the **DDS_LIVELINESS_CHANGED_STATUS** (p. 542) communication status.

This also resets the status so that it is no longer considered changed.

Parameters

status	<<***inout***>> *(p. 433)* **DDS_LivelinessChangedStatus** (p. 1176) to be filled in.

Exceptions

One	of the **Standard Return Codes** (p. 532)

9.245.2.20 get_requested_deadline_missed_status()

```
virtual DDS_ReturnCode_t DDSDataReader::get_requested_deadline_missed_status (
            DDS_RequestedDeadlineMissedStatus & status ) [inline], [virtual]
```

Accesses the **DDS_REQUESTED_DEADLINE_MISSED_STATUS** (p. 540) communication status.

This also resets the status so that it is no longer considered changed.

Parameters

status	<<***inout***>> *(p. 433)* **DDS_RequestedDeadlineMissedStatus** (p. 1306) to be filled in.

Exceptions

One	of the **Standard Return Codes** (p. 532)

9.245.2.21 get_requested_incompatible_qos_status()

```
virtual DDS_ReturnCode_t DDSDataReader::get_requested_incompatible_qos_status (
            DDS_RequestedIncompatibleQosStatus & status ) [inline], [virtual]
```

Accesses the **DDS_REQUESTED_INCOMPATIBLE_QOS_STATUS** (p. 540) communication status.

This also resets the status so that it is no longer considered changed.

Parameters

status	<<***inout***>> *(p. 433)* **DDS_RequestedIncompatibleQosStatus** (p. 1307) to be filled in.

Exceptions

One	of the **Standard Return Codes** (p. 532)

9.245.2.22 get_sample_lost_status()

```
virtual DDS_ReturnCode_t DDSDataReader::get_sample_lost_status (
            DDS_SampleLostStatus & status ) [inline], [virtual]
```

Accesses the **DDS_SAMPLE_LOST_STATUS** (p. 541) communication status.

This also resets the status so that it is no longer considered changed.

Parameters

status	<<***inout***>> *(p. 433)* **DDS_SampleLostStatus** (p. 1355) to be filled in.

Exceptions

One	of the **Standard Return Codes** (p. 532)

9.245.2.23 get_subscription_matched_status()

```
virtual DDS_ReturnCode_t DDSDataReader::get_subscription_matched_status (
            DDS_SubscriptionMatchedStatus & status ) [inline], [virtual]
```

Accesses the **DDS_SUBSCRIPTION_MATCHED_STATUS** (p. 543) communication status.

Parameters

status	<<***inout***>> *(p. 433)* **DDS_SubscriptionMatchedStatus** (p. 1379) to be filled in.

Exceptions

One	of the **Standard Return Codes** (p. 532)

9.245.2.24 get_datareader_cache_status()

```
virtual DDS_ReturnCode_t DDSDataReader::get_datareader_cache_status (
            DDS_DataReaderCacheStatus & status ) [inline], [virtual]
```

<<***extension***>> *(p. 432)* Gets the datareader cache status for this reader.

Parameters

status	<<***inout***>> *(p. 433)* **DDS_DataReaderCacheStatus** (p. 836) to be filled in.

Exceptions

One	of the **Standard Return Codes** (p. 532) or **DDS_RETCODE_NOT_ENABLED** (p. 533).

9.245.2.25 get_datareader_protocol_status()

```
virtual DDS_ReturnCode_t DDSDataReader::get_datareader_protocol_status (
            DDS_DataReaderProtocolStatus & status ) [inline], [virtual]
```

<<***extension***>> *(p. 432)* Gets the datareader protocol status for this reader.

This also resets the status so that it is no longer considered changed.

Parameters

status	<<***inout***>> *(p. 433)* **DDS_DataReaderProtocolStatus** (p. 847) to be filled in.

Exceptions

One	of the **Standard Return Codes** (p. 532) or **DDS_RETCODE_NOT_ENABLED** (p. 533).

9.245.2.26 get_matched_publication_datareader_protocol_status()

```
virtual DDS_ReturnCode_t DDSDataReader::get_matched_publication_datareader_protocol_status (
            DDS_DataReaderProtocolStatus & status,
          const DDS_InstanceHandle_t & publication_handle ) [inline], [virtual]
```

*<<**extension**>> (p. 432)* Gets the datareader protocol status for this reader, per matched publication identified by the publication_handle.

This also resets the status so that it is no longer considered changed.

Note: Status for a remote entity is only kept while the entity is alive. Once a remote entity is no longer alive, its status is deleted.

Parameters

status	*<<**inout**>> (p. 433).* The information to be filled in on the associated publication. Cannot be NULL.
publication_handle	*<<**in**>> (p. 433).* Handle to a specific publication associated with the **DDSDataWriter** (p. 1589). . Must correspond to a publication currently associated with the **DDSDataReader** (p. 1556).

Exceptions

One	of the **Standard Return Codes** (p. 532) or **DDS_RETCODE_NOT_ENABLED** (p. 533)

9.245.2.27 set_qos()

```
virtual DDS_ReturnCode_t DDSDataReader::set_qos (
            const DDS_DataReaderQos & qos ) [inline], [virtual]
```

Sets the reader QoS.

Modifies the QoS of the **DDSDataReader** (p. 1556).

The **DDS_DataReaderQos::user_data** (p. 862), **DDS_DataReaderQos::deadline** (p. 861), **DDS_DataReaderQos::latency_budget** (p. 861), **DDS_DataReaderQos::time_based_filter** (p. 863), **DDS_DataReaderQos::reader_data_lifecycle** (p. 863) can be changed. The other policies are immutable.

Parameters

qos	*<<**in**>> (p. 433)* The **DDS_DataReaderQos** (p. 858) to be set to. Policies must be consistent. Immutable policies cannot be changed after **DDSDataReader** (p. 1556) is enabled. The special value **DDS_DATAREADER_QOS_DEFAULT** (p. 136) can be used to indicate that the QoS of the **DDSDataReader** (p. 1556) should be changed to match the current default **DDS_DataReaderQos** (p. 858) set in the **DDSSubscriber** (p. 1860).

Exceptions

One	of the **Standard Return Codes** (p. 532), **DDS_RETCODE_IMMUTABLE_POLICY** (p. 533), or **DDS_RETCODE_INCONSISTENT_POLICY** (p. 533).

See also

DDS_DataReaderQos (p. 858) for rules on consistency among QoS

set_qos (abstract) (p. 1732)

DDSDataReader::set_qos (p. 1574)

Operations Allowed in Listener Callbacks (p. 1796)

9.245.2.28 get_qos()

```
virtual DDS_ReturnCode_t DDSDataReader::get_qos (
            DDS_DataReaderQos & qos ) [inline], [virtual]
```

Gets the reader QoS.

This method may potentially allocate memory depending on the sequences contained in some QoS policies.

Parameters

qos	<<***inout***>> *(p. 433)* The **DDS_DataReaderQos** (p. 858) to be filled up.

Exceptions

One	of the **Standard Return Codes** (p. 532)

See also

get_qos (abstract) (p. 1733)

9.245.2.29 set_qos_with_profile()

```
virtual DDS_ReturnCode_t DDSDataReader::set_qos_with_profile (
            const char * library_name,
            const char * profile_name ) [inline], [virtual]
```

<<***extension***>> *(p. 432)* Changes the QoS of this reader using the input XML QoS profile.

This operation modifies the QoS of the **DDSDataReader** (p. 1556).

The **DDS_DataReaderQos::user_data** (p. 862), **DDS_DataReaderQos::deadline** (p. 861), **DDS_DataReaderQos::latency_budget** (p. 861), **DDS_DataReaderQos::time_based_filter** (p. 863), **DDS_DataReaderQos::reader_data_lifecycle** (p. 863) can be changed. The other policies are immutable.

Parameters

library_name	*<<**in**>> (p. 433)* Library name containing the XML QoS profile. If library_name is null RTI Connext will use the default library (see **DDSSubscriber::set_default_library** (p. 1865)).
profile_name	*<<**in**>> (p. 433)* XML QoS Profile name. If profile_name is null RTI Connext will use the default profile (see **DDSSubscriber::set_default_profile** (p. 1866)).

Exceptions

One	of the **Standard Return Codes** (p. 532), **DDS_RETCODE_IMMUTABLE_POLICY** (p. 533), or **DDS_RETCODE_INCONSISTENT_POLICY** (p. 533).

See also

DDS_DataReaderQos (p. 858) for rules on consistency among QoS

DDSDataReader::set_qos (p. 1574)

Operations Allowed in Listener Callbacks (p. 1796)

9.245.2.30 set_property()

```
virtual  DDS_ReturnCode_t DDSDataReader::set_property (
             const char * property_name,
             const char * value,
             bool propagate )  [inline], [virtual]
```

Set the value for a property that applies to a DataReader.

Warning

This method is not implemented in all APIs and it's intended only for testing purposes. You should use **DDSData↩ Reader::set_qos** (p. 1574) instead.

Parameters

property_name	*<<**in**>> (p. 433).* Name of the property that you want to set.
value	*<<**in**>> (p. 433).* New value for the property.
propagate	*<<**in**>> (p. 433).* Indicates if the property will be propagated or not.

Exceptions

One	of the **Standard Return Codes** (p. 532)

See also

DDSDomainParticipant::set_property (p. 1676)

DDSDataWriter::set_property (p. 1605)

DDSDataReader::set_qos (p. 1574)

9.245.2.31 set_listener()

```
virtual DDS_ReturnCode_t DDSDataReader::set_listener (
            DDSDataReaderListener * l,
            DDS_StatusMask mask = DDS_STATUS_MASK_ALL ) [inline], [virtual]
```

Sets the reader listener.

Parameters

l	<<***in***>> *(p. 433)* **DDSDataReaderListener** (p. 1583) to set to
mask	<<***in***>> *(p. 433)* **DDS_StatusMask** (p. 537) associated with the **DDSDataReaderListener** (p. 1583).

Exceptions

One	of the **Standard Return Codes** (p. 532)

See also

set_listener (abstract) (p. 1733)

9.245.2.32 get_listener()

```
virtual DDSDataReaderListener * DDSDataReader::get_listener ( ) [inline], [virtual]
```

Gets the reader listener.

Returns

DDSDataReaderListener (p. 1583) of the **DDSDataReader** (p. 1556).

See also

get_listener (abstract) (p. 1734)

9.245.2.33 create_topic_query()

```
virtual DDSTopicQuery * DDSDataReader::create_topic_query (
            const DDS_TopicQuerySelection & selection ) [inline], [virtual]
```

Creates a **DDSTopicQuery** (p. 1895).

The returned **DDSTopicQuery** (p. 1895) will have been issued if the **DDSDataReader** (p. 1556) is enabled. Otherwise, the **DDSTopicQuery** (p. 1895) will be issued once the **DDSDataReader** (p. 1556) is enabled

Parameters

selection	<<***in***>> *(p. 433)* The selection with which to create the **DDSTopicQuery** (p. 1895). The special values **DDS_TOPIC_QUERY_SELECTION_SELECT_ALL** (p. 160) and **DDS_TOPIC_QUERY_SELECTION_USE_READER_CONTENT_FILTER** (p. 160) can be used. The expression can start with the special condition "@instance_state = ALIVE AND" followed by the rest of the expression. This restricts the selection to samples of alive instances.

Returns

return Created **DDSTopicQuery** (p. 1895). Returns NULL in case of failure.

9.245.2.34 delete_topic_query()

```
virtual DDS_ReturnCode_t DDSDataReader::delete_topic_query (
            DDSTopicQuery * query ) [inline], [virtual]
```

Deletes a **DDSTopicQuery** (p. 1895).

Cancels an active **DDSTopicQuery** (p. 1895). After deleting a TopicQuery, new DataWriters won't discover it and existing DataWriters currently publishing cached samples may stop before delivering all of them.

Parameters

query	<<***in***>> *(p. 433)* The **DDSTopicQuery** (p. 1895) to delete.

Exceptions

One	of the **Standard Return Codes** (p. 532)

9.245.2.35 lookup_topic_query()

```
virtual DDSTopicQuery * DDSDataReader::lookup_topic_query (
```

```
const DDS_GUID_t & guid ) [inline], [virtual]
```

Retrieves an existing **DDSTopicQuery** (p. 1895).

Retrieves the **DDSTopicQuery** (p. 1895) that corresponds to the input **DDS_GUID_t** (p. 1161).

If no TopicQuery is found for the specified GUID or the TopicQuery is marked for deletion, this returns NULL.

To get the **DDS_GUID_t** (p. 1161) associated with a **DDSTopicQuery** (p. 1895), use the method **DDSTopicQuery**↩ **::get_guid** (p. 1895).

Parameters

guid	<<***in***>> *(p. 433)* The **DDSTopicQuery** (p. 1895) GUID.

9.245.2.36 take_discovery_snapshot() [1/2]

```
virtual DDS_ReturnCode_t DDSDataReader::take_discovery_snapshot ( ) [inline], [virtual]
```

Take a snapshot of the compatible and incompatible remote writers matched by a local reader.

The snapshot will be printed through the **NDDSConfigLogger** (p. 2093). A possible output may be the following:

```
Remote writers that match the local reader domain=0 name="readerTestName"

guid="0x0101542A,0x2C59B595,0xA1693BDF:0x80000004"

topic="FooTopic" type="FooType"

------------------------------------------------------------------------

Compatible writers:

1. 0x0101D8D1,0x20B83C0D,0x4495246E:0x80000003 name="writer1TestName"

kind="unkeyed user datareader"

unicastLocators="udpv4://192.168.1.170:7411"

liveliness="ALIVE"

Incompatible writers:

1. 0x0101D8D1,0x20B83C0D,0x4495246E:0x80000103 name="writer2TestName"

kind="unkeyed user datareader"

unicastLocators="udpv4://192.168.1.170:7411"

reason="Inconsistent QoS"

------------------------------------------------------------------------
```

Exceptions

One	of the **Standard Return Codes** (p. 532).

9.245.2.37 take_discovery_snapshot() [2/2]

```
virtual DDS_ReturnCode_t DDSDataReader::take_discovery_snapshot (
            const char * file_name ) [inline], [virtual]
```

Take a snapshot of the compatible and incompatible remote writers matched by a local reader.

The snapshot will be printed in the file specified by file_name. A possible output may be the following:

```
Remote writers that match the local reader domain=0 name="readerTestName"

guid="0x0101542A,0x2C59B595,0xA1693BDF:0x80000004"

topic="FooTopic" type="FooType"

-------------------------------------------------------------------------

Compatible writers:

1. 0x0101D8D1,0x20B83C0D,0x4495246E:0x80000003 name="writer1TestName"

kind="unkeyed user datareader"

unicastLocators="udpv4://192.168.1.170:7411"

liveliness="ALIVE"

Incompatible writers:

1. 0x0101D8D1,0x20B83C0D,0x4495246E:0x80000103 name="writer2TestName"

kind="unkeyed user datareader"

unicastLocators="udpv4://192.168.1.170:7411"

reason="Inconsistent QoS"

-------------------------------------------------------------------------
```

Parameters

file_name	<<***in***>> *(*p. *433)* Name of the file where snapshot should be printed.

Exceptions

One	of the **Standard Return Codes** (p. 532).

9.245.2.38 enable()

```
virtual DDS_ReturnCode_t DDSDataReader::enable ( ) [inline], [virtual]
```

Enables the **DDSEntity** (p. 1731).

This operation enables the Entity. Entity objects can be created either enabled or disabled. This is controlled by the value of the **ENTITY_FACTORY** (p. 600) QoS policy on the corresponding factory for the **DDSEntity** (p. 1731).

By default, **ENTITY_FACTORY** (p. 600) is set so that it is not necessary to explicitly call **DDSEntity::enable** (p. 1734) on newly created entities.

The **DDSEntity::enable** (p. 1734) operation is idempotent. Calling enable on an already enabled Entity returns OK and has no effect.

If a **DDSEntity** (p. 1731) has not yet been enabled, the following kinds of operations may be invoked on it:

- set or get the QoS policies (including default QoS policies) and listener
- **DDSEntity::get_statuscondition** (p. 1735)
- 'factory' operations
- **DDSEntity::get_status_changes** (p. 1735) and other get status operations (although the status of a disabled entity never changes)
- 'lookup' operations

Other operations may explicitly state that they may be called on disabled entities; those that do not will return the error **DDS_RETCODE_NOT_ENABLED** (p. 533).

It is legal to delete an **DDSEntity** (p. 1731) that has not been enabled by calling the proper operation on its factory .

Entities created from a factory Entity that is disabled are created disabled, regardless of the setting of the **DDS_Entity↩ FactoryQosPolicy** (p. 1145).

Calling enable on an Entity whose factory Entity is not enabled will fail and return **DDS_RETCODE_PRECONDITION↩ _NOT_MET** (p. 532).

If **DDS_EntityFactoryQosPolicy::autoenable_created_entities** (p. 1147) is TRUE, the enable operation on a factory will automatically enable all entities created from that factory (for example, enabling a **DDSPublisher** (p. 1818) will enable all its contained **DDSDataWriter** (p. 1589) objects)

Listeners associated with an entity are not called until the entity is enabled.

Conditions associated with a disabled entity are "inactive," that is, they have a `trigger_value` == FALSE.

Exceptions

One	of the **Standard Return Codes** (p. 532), **Standard Return Codes** (p. 532) or **DDS_RETCODE_PRECONDITION_NOT_MET** (p. 532).

Implements **DDSEntity** (p. 1734).

9.245.2.39 get_statuscondition()

```
virtual DDSStatusCondition * DDSDataReader::get_statuscondition ( ) [inline], [virtual]
```

Allows access to the **DDSStatusCondition** (p. 1846) associated with the **DDSEntity** (p. 1731).

The returned condition can then be added to a **DDSWaitSet** (p. 1897) so that the application can wait for specific status changes that affect the **DDSEntity** (p. 1731).

Returns

the status condition associated with this entity.

Implements **DDSEntity** (p. 1735).

9.245.2.40 get_status_changes()

```
virtual DDS_StatusMask DDSDataReader::get_status_changes ( ) [inline], [virtual]
```

Retrieves the list of communication statuses in the **DDSEntity** (p. 1731) that are triggered.

That is, the list of statuses whose value has changed since the last time the application read the status using the get_*_status() method.

When the entity is first created or if the entity is not enabled, all communication statuses are in the "untriggered" state so the list returned by the get_status_changes operation will be empty.

The list of statuses returned by the get_status_changes operation refers to the status that are triggered on the Entity itself and does not include statuses that apply to contained entities.

Returns

list of communication statuses in the **DDSEntity** (p. 1731) that are triggered.

See also

Status Kinds (p. 533)

Implements **DDSEntity** (p. 1735).

9.245.2.41 get_instance_handle()

```
virtual DDS_InstanceHandle_t DDSDataReader::get_instance_handle ( ) [inline], [virtual]
```

Allows access to the **DDS_InstanceHandle_t** (p. 76) associated with the **DDSEntity** (p. 1731).

This operation returns the **DDS_InstanceHandle_t** (p. 76) that represents the **DDSEntity** (p. 1731).

Returns

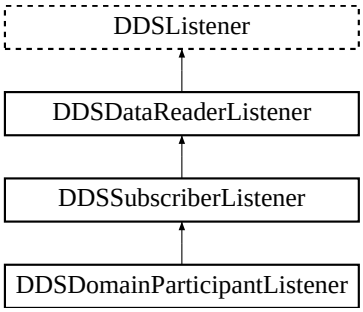
the instance handle associated with this entity.

Implements **DDSEntity** (p. 1736).

9.246 DDSDataReaderListener Class Reference

<<***interface***>> *(*p. *433)* **DDSListener** (p. 1794) for reader status.

Inheritance diagram for DDSDataReaderListener:



Public Member Functions

- virtual void **on_requested_deadline_missed** (**DDSDataReader** ∗reader, const **DDS_RequestedDeadline↩ MissedStatus** &status)

  Handles the ***DDS_REQUESTED_DEADLINE_MISSED_STATUS*** *(*p. *540) communication status.*
- virtual void **on_liveliness_changed** (**DDSDataReader** ∗reader, const **DDS_LivelinessChangedStatus** &status)

  Handles the ***DDS_LIVELINESS_CHANGED_STATUS*** *(*p. *542) communication status.*
- virtual void **on_requested_incompatible_qos** (**DDSDataReader** ∗reader, const **DDS_Requested↩ IncompatibleQosStatus** &status)

  Handles the ***DDS_REQUESTED_INCOMPATIBLE_QOS_STATUS*** *(*p. *540) communication status.*
- virtual void **on_sample_rejected** (**DDSDataReader** ∗reader, const **DDS_SampleRejectedStatus** &status)

  Handles the ***DDS_SAMPLE_REJECTED_STATUS*** *(*p. *541) communication status.*
- virtual void **on_data_available** (**DDSDataReader** ∗reader)

  Handle the ***DDS_DATA_AVAILABLE_STATUS*** *(*p. *541) communication status.*
- virtual void **on_sample_lost** (**DDSDataReader** ∗reader, const **DDS_SampleLostStatus** &status)

  Handles the ***DDS_SAMPLE_LOST_STATUS*** *(*p. *541) communication status.*
- virtual void **on_subscription_matched** (**DDSDataReader** ∗reader, const **DDS_SubscriptionMatchedStatus** &status)

  Handles the ***DDS_SUBSCRIPTION_MATCHED_STATUS*** *(*p. *543) communication status.*

9.246.1 Detailed Description

<<***interface***>> *(*p. *433)* **DDSListener** (p. 1794) for reader status.

Entity:

DDSDataReader (p. 1556)

Status:

DDS_DATA_AVAILABLE_STATUS (p. 541);
DDS_LIVELINESS_CHANGED_STATUS (p. 542), **DDS_LivelinessChangedStatus** (p. 1176);
DDS_REQUESTED_DEADLINE_MISSED_STATUS (p. 540), **DDS_RequestedDeadlineMissedStatus** (p. 1306);
DDS_REQUESTED_INCOMPATIBLE_QOS_STATUS (p. 540), **DDS_RequestedIncompatibleQosStatus** (p. 1307);
DDS_SAMPLE_LOST_STATUS (p. 541), **DDS_SampleLostStatus** (p. 1355);
DDS_SAMPLE_REJECTED_STATUS (p. 541), **DDS_SampleRejectedStatus** (p. 1356);
DDS_SUBSCRIPTION_MATCHED_STATUS (p. 543), **DDS_SubscriptionMatchedStatus** (p. 1379);

See also

Status Kinds (p. 533)

Operations Allowed in Listener Callbacks (p. 1796)

9.246.2 Member Function Documentation

9.246.2.1 on_requested_deadline_missed()

```
virtual void DDSDataReaderListener::on_requested_deadline_missed (
              DDSDataReader * reader,
             const  DDS_RequestedDeadlineMissedStatus & status )  [virtual]
```

Handles the **DDS_REQUESTED_DEADLINE_MISSED_STATUS** (p. 540) communication status.

9.246.2.2 on_liveliness_changed()

```
virtual void DDSDataReaderListener::on_liveliness_changed (
              DDSDataReader * reader,
             const  DDS_LivelinessChangedStatus & status )  [virtual]
```

Handles the **DDS_LIVELINESS_CHANGED_STATUS** (p. 542) communication status.

9.246.2.3 on_requested_incompatible_qos()

```
virtual void DDSDataReaderListener::on_requested_incompatible_qos (
              DDSDataReader * reader,
             const  DDS_RequestedIncompatibleQosStatus & status )  [virtual]
```

Handles the **DDS_REQUESTED_INCOMPATIBLE_QOS_STATUS** (p. 540) communication status.

9.246.2.4 on_sample_rejected()

```
virtual void DDSDataReaderListener::on_sample_rejected (
              DDSDataReader * reader,
             const  DDS_SampleRejectedStatus & status )  [virtual]
```

Handles the **DDS_SAMPLE_REJECTED_STATUS** (p. 541) communication status.

9.246.2.5 on_data_available()

```
virtual void DDSDataReaderListener::on_data_available (
              DDSDataReader * reader )  [virtual]
```

Handle the **DDS_DATA_AVAILABLE_STATUS** (p. 541) communication status.

9.246.2.6 on_sample_lost()

```
virtual void DDSDataReaderListener::on_sample_lost (
              DDSDataReader * reader,
             const  DDS_SampleLostStatus & status )  [virtual]
```

Handles the **DDS_SAMPLE_LOST_STATUS** (p. 541) communication status.

9.246.2.7 on_subscription_matched()

```
virtual void DDSDataReaderListener::on_subscription_matched (
              DDSDataReader * reader,
             const  DDS_SubscriptionMatchedStatus & status )  [virtual]
```

Handles the **DDS_SUBSCRIPTION_MATCHED_STATUS** (p. 543) communication status.

9.247 DDSDataReaderSeq Class Reference

Declares IDL `sequence` < **DDSDataReader** (p. 1556) > .

9.247.1 Detailed Description

Declares IDL `sequence` < **DDSDataReader** (p. 1556) > .

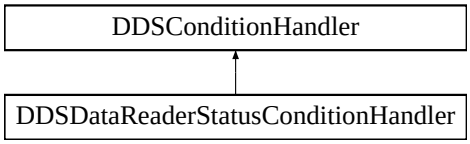
See also

FooSeq (p. 1965)

9.248 DDSDataReaderStatusConditionHandler Class Reference

<<***interface***>> *(p. 433)* Realization of a **DDSConditionHandler** (p. 1546) that handles the status of a **DDSData↩ Reader** (p. 1556).

Inheritance diagram for DDSDataReaderStatusConditionHandler:



Public Member Functions

- virtual void **on_condition_triggered** (**DDSCondition** ∗condition)

  *Handles the dispatch of a **DDSCondition** (p. 1544).*
- **DDSDataReaderStatusConditionHandler** (**DDSDataReader** ∗reader, **DDSDataReaderListener** ∗listener, **DDS_StatusMask** listener_mask)

  *Creates a new **DDSDataReaderStatusConditionHandler** (p. 1586) instance.*
- virtual ∼**DDSDataReaderStatusConditionHandler** ()

  *Deletes a **DDSDataReaderStatusConditionHandler** (p. 1586) instance previously created with **DDSDataReader↩ StatusConditionHandler::DDSDataReaderStatusConditionHandler** (p. 1587).*

9.248.1 Detailed Description

<<***interface***>> *(p. 433)* Realization of a **DDSConditionHandler** (p. 1546) that handles the status of a **DDSData↩ Reader** (p. 1556).

A **DDSDataReaderStatusConditionHandler** (p. 1586) demultiplexes a **DDSDataReader** (p. 1556) status change into the corresponding callback of a provided **DDSDataReaderListener** (p. 1583) implementation.

Note that the **DDSDataReaderListener** (p. 1583) notifications have different considerations than if the were made by the **DDSDataReader** (p. 1556) directly:

- Context: The **DDSDataReaderListener** (p. 1583) callback context is the one of the thread that dispatches the **DDSCondition** (p. 1544) where this handler is set. For instance, if you attach the condition to a **DDSAsync↩ WaitSet** (p. 1528), the context will be one of the threads within the pool.

- Status clearing: All the **DDSDataReader** (p. 1556)'s enabled statuses are cleared upon condition dispatch except the **DDS_DATA_AVAILABLE_STATUS** (p. 541), which will not be cleared until your application reads the data.
- Exclusive Area: Restrictions depend on the context of the dipsatching thread. For instance, if the **DDSAsyncWaitSet** (p. 1528) dispatches the condition, the listener notications are free of any exclusive area restrictions.

The **DDSDataReaderStatusConditionHandler** (p. 1586) is a convenience to handle the status changes of a **DDSDataReader** (p. 1556). You can install a **DDSDataReaderStatusConditionHandler** (p. 1586) as the handler of a reader's **DDSStatusCondition** (p. 1846). You can then attach it to a **DDSWaitSet** (p. 1897) or **DDSAsyncWaitSet** (p. 1528) and receive status changes notifications through a specific **DDSDataReaderListener** (p. 1583) implementation instance.

9.248.2 Constructor & Destructor Documentation

9.248.2.1 DDSDataReaderStatusConditionHandler()

```
DDSDataReaderStatusConditionHandler::DDSDataReaderStatusConditionHandler (
            DDSDataReader * reader,
            DDSDataReaderListener * listener,
            DDS_StatusMask listener_mask )
```

Creates a new **DDSDataReaderStatusConditionHandler** (p. 1586) instance.

The created DataReaderStatusConditionHandler can set as **DDSConditionHandler** (p. 1546) in any **DDSCondition** (p. 1544) and will demultiplex the specified status changes from the specified **DDSDataReader** (p. 1556)

Parameters

reader	<<***in***>> *(p. 433)* The **DDSDataReader** (p. 1556) for which the status changes are demultiplexed to the specified `listener`
listener	<<***in***>> *(p. 433)* that receives the status changes notifications from the specified `reader`.
listener_mask	<<***in***>> *(p. 433)* Specifies which status changes from `the` reader to demultiplex to the `listener`.

See also

DDSDataReader (p. 1556)

DDSDataReader::set_listener (p. 1577)

DDSStatusCondition (p. 1846)

DDSConditionHandler (p. 1546)

9.248.2.2 ~DDSDataReaderStatusConditionHandler()

```
virtual DDSDataReaderStatusConditionHandler::~DDSDataReaderStatusConditionHandler ( )  [virtual]
```

Deletes a **DDSDataReaderStatusConditionHandler** (p. 1586) instance previously created with **DDSDataReader↩ StatusConditionHandler::DDSDataReaderStatusConditionHandler** (p. 1587).

9.248.3 Member Function Documentation

9.248.3.1 on_condition_triggered()

```
virtual void DDSDataReaderStatusConditionHandler::on_condition_triggered (
              DDSCondition * condition )  [virtual]
```

Handles the dispatch of a **DDSCondition** (p. 1544).

This callback is called by **DDSCondition::dispatch** (p. 1546).

Implements **DDSConditionHandler** (p. 1547).

9.249 DDSDataTagQosPolicyHelper Class Reference

Policy helpers that facilitate management of the data tags in the input policy.

Static Public Member Functions

- static **DDS_Long get_number_of_tags** (**DDS_DataTagQosPolicy** &policy)

  Gets the number of data tags in the input policy.
- static **DDS_ReturnCode_t assert_tag** (**DDS_DataTagQosPolicy** &policy, const char ∗name, const char ∗value)

  Asserts the tag identified by name in the input policy.
- static **DDS_ReturnCode_t add_tag** (**DDS_DataTagQosPolicy** &policy, const char ∗name, const char ∗value)

  Adds a new tag to the input policy.
- static struct **DDS_Tag** ∗ **lookup_tag** (**DDS_DataTagQosPolicy** &policy, const char ∗name)

  Searches by tag name for a tag in the input policy.
- static **DDS_ReturnCode_t remove_tag** (**DDS_DataTagQosPolicy** &policy, const char ∗name)

  Removes a tag from the input policy.

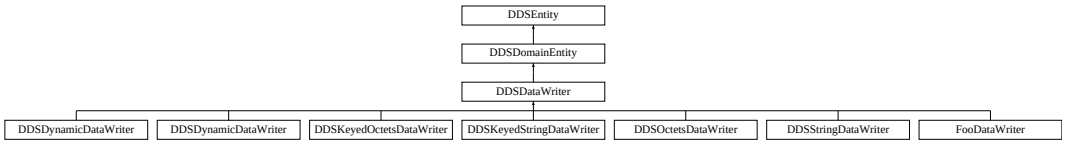
9.249.1 Detailed Description

Policy helpers that facilitate management of the data tags in the input policy.

9.250 DDSDataWriter Class Reference

<<***interface***>> *(*p. *433)* Allows an application to set the value of the data to be published under a given **DDSTopic** (p. 1885).

Inheritance diagram for DDSDataWriter:



Public Member Functions

- virtual **DDS_ReturnCode_t get_liveliness_lost_status** (**DDS_LivelinessLostStatus** &status)

  Accesses the ***DDS_LIVELINESS_LOST_STATUS*** *(*p. *542) communication status.*
- virtual **DDS_ReturnCode_t get_offered_deadline_missed_status** (**DDS_OfferedDeadlineMissedStatus** &status)

  Accesses the ***DDS_OFFERED_DEADLINE_MISSED_STATUS*** *(*p. *539) communication status.*
- virtual **DDS_ReturnCode_t get_offered_incompatible_qos_status** (**DDS_OfferedIncompatibleQosStatus** &status)

  Accesses the ***DDS_OFFERED_INCOMPATIBLE_QOS_STATUS*** *(*p. *540) communication status.*
- virtual **DDS_ReturnCode_t get_publication_matched_status** (**DDS_PublicationMatchedStatus** &status)

  Accesses the ***DDS_PUBLICATION_MATCHED_STATUS*** *(*p. *543) communication status.*
- virtual **DDS_ReturnCode_t get_reliable_writer_cache_changed_status** (**DDS_ReliableWriterCache↩ ChangedStatus** &status)

  <<***extension***>> *(*p. *432) Get the reliable cache status for this writer.*
- virtual **DDS_ReturnCode_t get_reliable_reader_activity_changed_status** (**DDS_ReliableReaderActivity↩ ChangedStatus** &status)

  <<***extension***>> *(*p. *432) Get the reliable reader activity changed status for this writer.*
- virtual **DDS_ReturnCode_t get_datawriter_cache_status** (**DDS_DataWriterCacheStatus** &status)

  <<***extension***>> *(*p. *432) Get the datawriter cache status for this writer.*
- virtual **DDS_ReturnCode_t get_datawriter_protocol_status** (**DDS_DataWriterProtocolStatus** &status)

  <<***extension***>> *(*p. *432) Get the datawriter protocol status for this writer.*
- virtual **DDS_ReturnCode_t get_matched_subscription_datawriter_protocol_status** (**DDS_DataWriter↩ ProtocolStatus** &status, const **DDS_InstanceHandle_t** &subscription_handle)

  <<***extension***>> *(*p. *432) Get the datawriter protocol status for this writer, per matched subscription identified by the subscription_handle.*
- virtual **DDS_ReturnCode_t is_matched_subscription_active** (**DDS_Boolean** &is_active, const **DDS_↩ InstanceHandle_t** &subscription_handle)

  Check if a subscription currently matched with a DataWriter is active.
- virtual **DDS_ReturnCode_t get_service_request_accepted_status** (**DDS_ServiceRequestAcceptedStatus** &status)

  Accesses the ***DDS_SERVICE_REQUEST_ACCEPTED_STATUS*** *(*p. *544) communication status.*
- virtual **DDS_ReturnCode_t get_matched_subscription_datawriter_protocol_status_by_locator** (**DDS_↩ DataWriterProtocolStatus** &status, const **DDS_Locator_t** &locator)

*<<**extension**>> (p. 432) Get the datawriter protocol status for this writer, per matched subscription identified by the locator.*

- virtual **DDS_ReturnCode_t assert_liveliness** ()

  *This operation manually asserts the liveliness of this **DDSDataWriter** (p. 1589).*

- virtual **DDS_ReturnCode_t get_matched_subscription_locators** (**DDS_LocatorSeq** &locators)

  *<<**extension**>> (p. 432) Retrieve the list of locators for subscriptions currently "associated" with this **DDSDataWriter** (p. 1589).*

- virtual **DDS_ReturnCode_t get_matched_subscriptions** (**DDS_InstanceHandleSeq** &subscription_handles)

  *Retrieve the list of subscriptions currently "associated" with this **DDSDataWriter** (p. 1589).*

- virtual **DDS_ReturnCode_t get_matched_subscription_data** (**DDS_SubscriptionBuiltinTopicData** &subscription_data, const **DDS_InstanceHandle_t** &subscription_handle)

  *This operation retrieves the information on a subscription that is currently "associated" with the **DDSDataWriter** (p. 1589).*

- virtual **DDS_ReturnCode_t get_matched_subscription_participant_data** (**DDS_ParticipantBuiltinTopic↩ Data** &participant_data, const **DDS_InstanceHandle_t** &subscription_handle)

  *This operation retrieves the information on the discovered **DDSDomainParticipant** (p. 1619) associated with the subscription that is currently matching with the **DDSDataWriter** (p. 1589).*

- virtual **DDSTopic** ∗ **get_topic** ()

  *This operation returns the **DDSTopic** (p. 1885) associated with the **DDSDataWriter** (p. 1589).*

- virtual **DDSPublisher** ∗ **get_publisher** ()

  *This operation returns the **DDSPublisher** (p. 1818) to which the **DDSDataWriter** (p. 1589) belongs.*

- virtual **DDS_ReturnCode_t wait_for_acknowledgments** (const **DDS_Duration_t** &max_wait)

  *Blocks the calling thread until all data written by reliable **DDSDataWriter** (p. 1589) entity is acknowledged, or until timeout expires.*

- virtual **DDS_ReturnCode_t is_sample_app_acknowledged** (**DDS_Boolean** &is_app_ack, const struct **DDS_SampleIdentity_t** &identity)

  This method can be used to see if a sample has been application acknowledged.

- virtual **DDS_ReturnCode_t wait_for_asynchronous_publishing** (const **DDS_Duration_t** &max_wait)

  *<<**extension**>> (p. 432) Blocks the calling thread until asynchronous sending is complete.*

- virtual **DDS_ReturnCode_t set_qos** (const **DDS_DataWriterQos** &qos)

  Sets the writer QoS.

- virtual **DDS_ReturnCode_t set_property** (const char ∗property_name, const char ∗value, bool propagate)

  Set the value for a property that applies to a DataWriter.

- virtual **DDS_ReturnCode_t set_qos_with_profile** (const char ∗library_name, const char ∗profile_name)

  *<<**extension**>> (p. 432) Change the QoS of this writer using the input XML QoS profile.*

- virtual **DDS_ReturnCode_t get_qos** (**DDS_DataWriterQos** &qos)

  Gets the writer QoS.

- virtual **DDS_ReturnCode_t set_listener** (**DDSDataWriterListener** ∗l, **DDS_StatusMask** mask= **DDS_↩ STATUS_MASK_ALL**)

  Sets the writer listener.

- virtual **DDSDataWriterListener** ∗ **get_listener** ()

  Get the writer listener.

- virtual **DDS_ReturnCode_t flush** ()

  *<<**extension**>> (p. 432) Flushes the batch in progress in the context of the calling thread.*

- virtual **DDS_ReturnCode_t take_discovery_snapshot** ()

  Take a snapshot of the compatible and incompatible remote readers matched by a local writer.

- virtual **DDS_ReturnCode_t take_discovery_snapshot** (const char ∗file_name)

  Take a snapshot of the compatible and incompatible remote readers matched by a local writer.

- virtual **DDS_ReturnCode_t enable** ()

*Enables the **DDSEntity** (p. 1731).*

- virtual **DDSStatusCondition** * **get_statuscondition** ()

  *Allows access to the **DDSStatusCondition** (p. 1846) associated with the **DDSEntity** (p. 1731).*

- virtual **DDS_StatusMask** **get_status_changes** ()

  *Retrieves the list of communication statuses in the **DDSEntity** (p. 1731) that are triggered.*

- virtual **DDS_InstanceHandle_t** **get_instance_handle** ()

  *Allows access to the **DDS_InstanceHandle_t** (p. 76) associated with the **DDSEntity** (p. 1731).*

9.250.1 Detailed Description

<<***interface***>> *(p. 433)* Allows an application to set the value of the data to be published under a given **DDSTopic** (p. 1885).

QoS:

DDS_DataWriterQos (p. 909)

Status:

DDS_LIVELINESS_LOST_STATUS (p. 542), **DDS_LivelinessLostStatus** (p. 1179);
DDS_OFFERED_DEADLINE_MISSED_STATUS (p. 539), **DDS_OfferedDeadlineMissedStatus** (p. 1219);
DDS_OFFERED_INCOMPATIBLE_QOS_STATUS (p. 540), **DDS_OfferedIncompatibleQosStatus** (p. 1221);
DDS_PUBLICATION_MATCHED_STATUS (p. 543), **DDS_PublicationMatchedStatus** (p. 1273);
DDS_RELIABLE_READER_ACTIVITY_CHANGED_STATUS (p. 545), **DDS_ReliableReaderActivityChangedStatus** (p. 1300);
DDS_RELIABLE_WRITER_CACHE_CHANGED_STATUS (p. 545), **DDS_ReliableWriterCacheChangedStatus** (p. 1302).

Listener:

DDSDataWriterListener (p. 1612)

A **DDSDataWriter** (p. 1589) is attached to exactly one **DDSPublisher** (p. 1818), that acts as a factory for it.

A **DDSDataWriter** (p. 1589) is bound to exactly one **DDSTopic** (p. 1885) and therefore to exactly one data type. The **DDSTopic** (p. 1885) must exist prior to the **DDSDataWriter** (p. 1589)'s creation.

DDSDataWriter (p. 1589) is an abstract class. It must be specialized for each particular application data-type (see **USER_DATA** (p. 653)). The additional methods or functions that must be defined in the auto-generated class for a hypothetical application type **Foo** (p. 1916) are specified in the example type **DDSDataWriter** (p. 1589).

The following operations may be called even if the **DDSDataWriter** (p. 1589) is not enabled. Other operations will fail with **DDS_RETCODE_NOT_ENABLED** (p. 533) if called on a disabled **DDSDataWriter** (p. 1589):

- The base-class operations **DDSDataWriter::set_qos** (p. 1604), **DDSDataWriter::get_qos** (p. 1606), **DDSDataWriter::set_listener** (p. 1607), **DDSDataWriter::get_listener** (p. 1607), **DDSEntity::enable** (p. 1734), **DDSEntity::get_statuscondition** (p. 1735) and **DDSEntity::get_status_changes** (p. 1735)

- **DDSDataWriter::get_liveliness_lost_status** (p. 1592), **DDSDataWriter::get_offered_deadline_missed_status** (p. 1592), **DDSDataWriter::get_offered_incompatible_qos_status** (p. 1593), **DDSDataWriter::get_publication_matched_status** (p. 1593), **DDSDataWriter::get_reliable_writer_cache_changed_status** (p. 1594), **DDSDataWriter::get_reliable_reader_activity_changed_status** (p. 1594) **DDSDataWriter::get_service_request_accepted_status** (p. 1596)

Several **DDSDataWriter** (p. 1589) may operate in different threads. If they share the same **DDSPublisher** (p. 1818), the middleware guarantees that its operations are thread-safe.

See also

FooDataWriter (p. 1944)

Operations Allowed in Listener Callbacks (p. 1796)

Examples

HelloWorldSupport.cxx, and **HelloWorld_publisher.cxx**.

9.250.2 Member Function Documentation

9.250.2.1 get_liveliness_lost_status()

```
virtual DDS_ReturnCode_t DDSDataWriter::get_liveliness_lost_status (
            DDS_LivelinessLostStatus & status ) [inline], [virtual]
```

Accesses the **DDS_LIVELINESS_LOST_STATUS** (p. 542) communication status.

This also resets the status so that it is no longer considered changed.

Parameters

status	<<***inout***>> *(p. 433)* **DDS_LivelinessLostStatus** (p. 1179) to be filled in.

Exceptions

One	of the **Standard Return Codes** (p. 532)

9.250.2.2 get_offered_deadline_missed_status()

```
virtual DDS_ReturnCode_t DDSDataWriter::get_offered_deadline_missed_status (
            DDS_OfferedDeadlineMissedStatus & status ) [inline], [virtual]
```

Accesses the **DDS_OFFERED_DEADLINE_MISSED_STATUS** (p. 539) communication status.

This also resets the status so that it is no longer considered changed.

Parameters

status	<<***inout***>> *(p. 433)* **DDS_OfferedDeadlineMissedStatus** (p. 1219) to be filled in.

Exceptions

One	of the **Standard Return Codes** (p. 532)

9.250.2.3 get_offered_incompatible_qos_status()

```
virtual DDS_ReturnCode_t DDSDataWriter::get_offered_incompatible_qos_status (
            DDS_OfferedIncompatibleQosStatus & status ) [inline], [virtual]
```

Accesses the **DDS_OFFERED_INCOMPATIBLE_QOS_STATUS** (p. 540) communication status.

This also resets the status so that it is no longer considered changed.

Parameters

status	<<***inout***>> *(p. 433)* **DDS_OfferedIncompatibleQosStatus** (p. 1221) to be filled in.

Exceptions

One	of the **Standard Return Codes** (p. 532)

9.250.2.4 get_publication_matched_status()

```
virtual DDS_ReturnCode_t DDSDataWriter::get_publication_matched_status (
            DDS_PublicationMatchedStatus & status ) [inline], [virtual]
```

Accesses the **DDS_PUBLICATION_MATCHED_STATUS** (p. 543) communication status.

This also resets the status so that it is no longer considered changed.

Parameters

status	<<***inout***>> *(p. 433)* **DDS_PublicationMatchedStatus** (p. 1273) to be filled in.

Exceptions

One	of the **Standard Return Codes** (p. 532)

9.250.2.5 get_reliable_writer_cache_changed_status()

```
virtual DDS_ReturnCode_t DDSDataWriter::get_reliable_writer_cache_changed_status (
            DDS_ReliableWriterCacheChangedStatus & status ) [inline], [virtual]
```

<<***extension***>> *(p. 432)* Get the reliable cache status for this writer.

This also resets the status so that it is no longer considered changed.

Parameters

status	<<***inout***>> *(p. 433)* **DDS_ReliableWriterCacheChangedStatus** (p. 1302) to be filled in.

Exceptions

One	of the **Standard Return Codes** (p. 532)

9.250.2.6 get_reliable_reader_activity_changed_status()

```
virtual DDS_ReturnCode_t DDSDataWriter::get_reliable_reader_activity_changed_status (
            DDS_ReliableReaderActivityChangedStatus & status ) [inline], [virtual]
```

<<***extension***>> *(p. 432)* Get the reliable reader activity changed status for this writer.

This also resets the status so that it is no longer considered changed.

Parameters

status	<<***inout***>> *(p. 433)* **DDS_ReliableReaderActivityChangedStatus** (p. 1300) to be filled in.

Exceptions

One	of the **Standard Return Codes** (p. 532)

9.250.2.7 get_datawriter_cache_status()

```
virtual DDS_ReturnCode_t DDSDataWriter::get_datawriter_cache_status (
            DDS_DataWriterCacheStatus & status ) [inline], [virtual]
```

<<***extension***>> *(p. 432)* Get the datawriter cache status for this writer.

This also resets the status so that it is no longer considered changed.

Parameters

status	<<***inout***>> *(p. 433)* **DDS_DataWriterCacheStatus** (p. 889) to be filled in.

Exceptions

One	of the **Standard Return Codes** (p. 532) or **DDS_RETCODE_NOT_ENABLED** (p. 533).

9.250.2.8 get_datawriter_protocol_status()

```
virtual DDS_ReturnCode_t DDSDataWriter::get_datawriter_protocol_status (
            DDS_DataWriterProtocolStatus & status ) [inline], [virtual]
```

<<***extension***>> *(p. 432)* Get the datawriter protocol status for this writer.

This also resets the status so that it is no longer considered changed.

Parameters

status	<<***inout***>> *(p. 433)* **DDS_DataWriterProtocolStatus** (p. 897) to be filled in.

Exceptions

One	of the **Standard Return Codes** (p. 532) or **DDS_RETCODE_NOT_ENABLED** (p. 533).

9.250.2.9 get_matched_subscription_datawriter_protocol_status()

```
virtual DDS_ReturnCode_t DDSDataWriter::get_matched_subscription_datawriter_protocol_status (
            DDS_DataWriterProtocolStatus & status,
          const DDS_InstanceHandle_t & subscription_handle ) [inline], [virtual]
```

*<<**extension**>> (p. 432)* Get the datawriter protocol status for this writer, per matched subscription identified by the subscription_handle.

This also resets the status so that it is no longer considered changed.

Note: Status for a remote entity is only kept while the entity is alive. Once a remote entity is no longer alive, its status is deleted.

Parameters

status	*<<**inout**>> (p. 433)* **DDS_DataWriterProtocolStatus** (p. 897) to be filled in.
subscription_handle	*<<**in**>> (p. 433)* Handle to a specific subscription associated with the **DDSDataReader** (p. 1556). Must correspond to a subscription currently associated with the **DDSDataWriter** (p. 1589).

Exceptions

One	of the **Standard Return Codes** (p. 532) or **DDS_RETCODE_NOT_ENABLED** (p. 533).

9.250.2.10 is_matched_subscription_active()

```
virtual DDS_ReturnCode_t DDSDataWriter::is_matched_subscription_active (
            DDS_Boolean & is_active,
          const DDS_InstanceHandle_t & subscription_handle ) [inline], [virtual]
```

Check if a subscription currently matched with a DataWriter is active.

This API is used for querying the endpoint liveliness of a matched subscription. A matched subscription will be marked as inactive when it becomes nonprogessing (e.g., not responding to a DataWriter's heartbeats, or letting its internal queue fill up without taking the available data). Note that if the participant associated with the matched subscription loses liveliness, the **DDS_InstanceHandle_t** (p. 76) will become invalid and this function will fail with **DDS_RETCODE_↩ BAD_PARAMETER** (p. 532).

Parameters

subscription_handle	*<<**in**>> (p. 433)* The **DDS_InstanceHandle_t** (p. 76) of the matched subscription. See **DDSDataWriter::get_matched_subscriptions** (p. 1599) for a description of what is considered a matched subscription.
is_active	*<<**out**>> (p. 433)* Indicates whether or not the matched subscription is active.

Exceptions

One	of the **Standard Return Codes** (p. 532)

9.250.2.11 get_service_request_accepted_status()

```
virtual DDS_ReturnCode_t DDSDataWriter::get_service_request_accepted_status (
            DDS_ServiceRequestAcceptedStatus & status ) [inline], [virtual]
```

Accesses the **DDS_SERVICE_REQUEST_ACCEPTED_STATUS** (p. 544) communication status.

This also resets the status so that it is no longer considered changed.

Parameters

status	<<***inout***>> *(p. 433)* **DDS_ServiceRequestAcceptedStatus** (p. 1361) to be filled in.

Exceptions

One	of the **Standard Return Codes** (p. 532)

9.250.2.12 get_matched_subscription_datawriter_protocol_status_by_locator()

```
virtual DDS_ReturnCode_t DDSDataWriter::get_matched_subscription_datawriter_protocol_status_by_↩
locator (
             DDS_DataWriterProtocolStatus & status,
            const DDS_Locator_t & locator ) [inline], [virtual]
```

<<***extension***>> *(p. 432)* Get the datawriter protocol status for this writer, per matched subscription identified by the locator.

This also resets the status so that it is no longer considered changed.

Note: Status for a remote entity is only kept while the entity is alive. Once a remote entity is no longer alive, its status is deleted.

Parameters

status	<<***inout***>> *(p. 433)* **DDS_DataWriterProtocolStatus** (p. 897) to be filled in
locator	<<***in***>> *(p. 433)* Locator to a specific locator associated with the **DDSDataReader** (p. 1556). Must correspond to a locator of one or more subscriptions currently associated with the **DDSDataWriter** (p. 1589).

Exceptions

One	of the **Standard Return Codes** (p. 532) or **DDS_RETCODE_NOT_ENABLED** (p. 533).

9.250.2.13 assert_liveliness()

```
virtual DDS_ReturnCode_t DDSDataWriter::assert_liveliness ( ) [inline], [virtual]
```

This operation manually asserts the liveliness of this **DDSDataWriter** (p. 1589).

This is used in combination with the **LIVELINESS** (p. 607) policy to indicate to RTI Connext that the **DDSDataWriter** (p. 1589) remains active.

You only need to use this operation if the **LIVELINESS** (p. 607) setting is either **DDS_MANUAL_BY_PARTICIPANT_↩ LIVELINESS_QOS** (p. 608) or **DDS_MANUAL_BY_TOPIC_LIVELINESS_QOS** (p. 608). Otherwise, it has no effect.

Note: writing data via the **FooDataWriter::write** (p. 1951) or **FooDataWriter::write_w_timestamp** (p. 1955) operation asserts liveliness on the **DDSDataWriter** (p. 1589) itself, and its **DDSDomainParticipant** (p. 1619). Consequently the use of **assert_liveliness()** (p. 1597) is only needed if the application is not writing data regularly.

Exceptions

One	of the **Standard Return Codes** (p. 532) or **DDS_RETCODE_NOT_ENABLED** (p. 533)

See also

DDS_LivelinessQosPolicy (p. 1180)

9.250.2.14 get_matched_subscription_locators()

```
virtual DDS_ReturnCode_t DDSDataWriter::get_matched_subscription_locators (
            DDS_LocatorSeq & locators ) [inline], [virtual]
```

*<<**extension**>> (p. 432)* Retrieve the list of locators for subscriptions currently "associated" with this **DDSDataWriter** (p. 1589).

The locators returned in the `locators` list are the ones that are used by the DDS implementation to communicate with the corresponding matched **DDSDataReader** (p. 1556) entities.

Parameters

locators	*<<**inout**>> (p. 433).* Handles of all the matched subscription locators.

The sequence will be grown if the sequence has ownership and the system has the corresponding resources. Use a sequence without ownership to avoid dynamic memory allocation. If the sequence is too small to store all the matches and the system can not resize the sequence, this method will fail with **DDS_RETCODE_OUT_OF_RESOURCES** (p. 533). .

Exceptions

One	of the **Standard Return Codes** (p. 532), or **DDS_RETCODE_OUT_OF_RESOURCES** (p. 533) if the sequence is too small and the system can not resize it, or **DDS_RETCODE_NOT_ENABLED** (p. 533)

9.250.2.15 get_matched_subscriptions()

```
virtual  DDS_ReturnCode_t DDSDataWriter::get_matched_subscriptions (
              DDS_InstanceHandleSeq & subscription_handles )  [inline], [virtual]
```

Retrieve the list of subscriptions currently "associated" with this **DDSDataWriter** (p. 1589).

A subscription is considered to be matching if all of the following criteria are true:

- The subscription is within the same domain as this publication.
- The subscription has a matching **DDSTopic** (p. 1885).
- The subscription has compatible QoS.
- If the applications are using partitions, the subscription shares a common partition with this publication.
- The **DDSDomainParticipant** (p. 1619) has not indicated that the subscription's **DDSDomainParticipant** (p. 1619) should be "ignored" by means of the **DDSDomainParticipant::ignore_publication** (p. 1664) API.
- If the publication is using the **DDS_MultiChannelQosPolicy** (p. 1214) and the subscription is using a **DDSContentFilteredTopic** (p. 1551), there is an intersection between at least one of the associated filters.
- If the endpoints need to exchange key material to communicate (i.e., they are securing their communications), the writer has completed the key exchange with reader.

The handles returned in the `subscription_handles` list are the ones that RTI Connext uses to locally identify the corresponding matched **DDSDataReader** (p. 1556) entities. These handles match the ones that appear in the **DDS_SampleInfo::instance_handle** (p. 1349) field of the **DDS_SampleInfo** (p. 1344) when reading the **DDS_SUBSCRIPTION_TOPIC_NAME** (p. 496) builtin topic.

This API may return the subscription handles of subscriptions that are inactive. **DDSDataWriter::is_matched_subscription_active** (p. 1596) can be used to check this.

Parameters

subscription_handles	<<***inout***>> *(p. 433).* The handles of all the matched subscriptions.

The sequence will be grown if the sequence has ownership and the system has the corresponding resources. Use a sequence without ownership to avoid dynamic memory allocation. If the sequence is too small to store all of the matches and the system cannot resize the sequence, this method will fail with **DDS_RETCODE_OUT_OF_RESOURCES** (p. 533).

The maximum number of matches possible is configured with **DDS_DomainParticipantResourceLimitsQosPolicy** (p. 977). You can use a zero-maximum sequence without ownership to quickly check whether there are any matches without allocating any memory. .

Exceptions

One	of the **Standard Return Codes** (p. 532), or **DDS_RETCODE_OUT_OF_RESOURCES** (p. 533) if the sequence is too small and the system cannot resize it, or **DDS_RETCODE_NOT_ENABLED** (p. 533)

9.250.2.16 get_matched_subscription_data()

```
virtual  DDS_ReturnCode_t DDSDataWriter::get_matched_subscription_data (
              DDS_SubscriptionBuiltinTopicData & subscription_data,
             const  DDS_InstanceHandle_t & subscription_handle )  [inline], [virtual]
```

This operation retrieves the information on a subscription that is currently "associated" with the **DDSDataWriter** (p. 1589).

The `subscription_handle` must correspond to a subscription currently associated with the **DDSDataWriter** (p. 1589). Otherwise, the operation will fail and fail with **DDS_RETCODE_BAD_PARAMETER** (p. 532). Use **DDSData↩ Writer::get_matched_subscriptions** (p. 1599) to find the subscriptions that are currently matched with the **DDSData↩ Writer** (p. 1589).

The above information is also available through **DDSDataReaderListener::on_data_available()** (p. 1585) (if a reader listener is installed on the **DDSSubscriptionBuiltinTopicDataDataReader** (p. 1882)).

When the subscription data is updated, for example when the content filter property changes, there is a small window of time in between when the DataWriter is made aware of these changes and when they actually take effect. Taking effect in this example means that the DataWriter will perform writer-side filtering using the new filter property values (filter expression and/or parameters).

When the DataWriter is made aware of the changes they will first be seen in the **DDSDataReaderListener::on_↩ data_available()** (p. 1585) of the **DDSSubscriptionBuiltinTopicDataDataReader** (p. 1882). When these changes are applied, they will be seen in the output of this API because this API blocks until the most recent changes known to the DataWriter have taken effect. This API will only block when called outside of a listener callback, in order to not block the internal threads from making progress.

If application behavior depends on being made aware of information about a subscription only after it has taken effect on the DataWriter, the recommended pattern for usage of this API is to wait for subscription data to be received either through polling this API or by installing a listener on the **DDSSubscriptionBuiltinTopicDataDataReader** (p. 1882). When a new sample is received by the builtin DataReader, this API may be called in a separate thread and will return the expected matched subscription data once it has been applied to the DataWriter.

Because this API blocks, it is possible for this API to time out while waiting for the changes to be applied. A timeout may happen if the DataReader's subscription data is changing rapidly, preventing the DataWriter from returning valid information before newer data has been received, or if an application is performing a task in a listener callback, thereby preventing the middleware's threads from executing events in a timely manner.

Note: This operation does not retrieve the **DDS_SubscriptionBuiltinTopicData::type_code** (p. 1376). This information is available through **DDSDataReaderListener::on_data_available()** (p. 1585) (if a reader listener is installed on the **DDSSubscriptionBuiltinTopicDataDataReader** (p. 1882)).

Parameters

subscription_data	*<<**inout**>> (p. 433).* The information to be filled in on the associated subscription. Cannot be NULL.
subscription_handle	*<<**in**>> (p. 433).* Handle to a specific subscription associated with the **DDSDataReader** (p. 1556). Must correspond to a subscription currently associated with the **DDSDataWriter** (p. 1589).

Exceptions

One	of the **Standard Return Codes** (p. 532), **DDS_RETCODE_NOT_ENABLED** (p. 533), or **DDS_RETCODE_TIMEOUT** (p. 533)

9.250.2.17 get_matched_subscription_participant_data()

```
virtual DDS_ReturnCode_t DDSDataWriter::get_matched_subscription_participant_data (
            DDS_ParticipantBuiltinTopicData & participant_data,
          const DDS_InstanceHandle_t & subscription_handle ) [inline], [virtual]
```

This operation retrieves the information on the discovered **DDSDomainParticipant** (p. 1619) associated with the subscription that is currently matching with the **DDSDataWriter** (p. 1589).

The `subscription_handle` must correspond to a subscription currently associated with the **DDSDataWriter** (p. 1589). Otherwise, the operation will fail with **DDS_RETCODE_BAD_PARAMETER** (p. 532). The operation may also fail with **DDS_RETCODE_PRECONDITION_NOT_MET** (p. 532) if the subscription corresponds to the same **DDSDomainParticipant** (p. 1619) that the DataWriter belongs to. Use **DDSDataWriter::get_matched_subscriptions** (p. 1599) to find the subscriptions that are currently matched with the **DDSDataWriter** (p. 1589).

Parameters

participant_data	*<<**inout**>> (p. 433).* The information to be filled in on the associated participant Cannot be NULL.
subscription_handle	*<<**in**>> (p. 433).* Handle to a specific subscription associated with the **DDSDataReader** (p. 1556). Must correspond to a subscription currently associated with the **DDSDataWriter** (p. 1589).

Exceptions

One	of the **Standard Return Codes** (p. 532), or **DDS_RETCODE_NOT_ENABLED** (p. 533)

9.250.2.18 get_topic()

```
virtual DDSTopic * DDSDataWriter::get_topic ( ) [inline], [virtual]
```

This operation returns the **DDSTopic** (p. 1885) associated with the **DDSDataWriter** (p. 1589).

This is the same **DDSTopic** (p. 1885) that was used to create the **DDSDataWriter** (p. 1589).

Returns

DDSTopic (p. 1885) that was used to create the **DDSDataWriter** (p. 1589).

9.250.2.19 get_publisher()

```
virtual DDSPublisher * DDSDataWriter::get_publisher ( ) [inline], [virtual]
```

This operation returns the **DDSPublisher** (p. 1818) to which the **DDSDataWriter** (p. 1589) belongs.

Returns

DDSPublisher (p. 1818) to which the **DDSDataWriter** (p. 1589) belongs.

9.250.2.20 wait_for_acknowledgments()

```
virtual DDS_ReturnCode_t DDSDataWriter::wait_for_acknowledgments (
            const DDS_Duration_t & max_wait ) [inline], [virtual]
```

Blocks the calling thread until all data written by reliable **DDSDataWriter** (p. 1589) entity is acknowledged, or until timeout expires.

This operation blocks the calling thread until either all data written by the reliable **DDSDataWriter** (p. 1589) entity is acknowledged by (a) all reliable **DDSDataReader** (p. 1556) entities that are matched and alive and (b) by all required subscriptions, or until the duration specified by the `max_wait` parameter elapses, whichever happens first. A successful completion indicates that all the samples written have been acknowledged by all reliable matched data readers and by all required subscriptions; a timeout indicates that max_wait elapsed before all the data was acknowledged.

Note that if a thread is blocked in the call to wait_for_acknowledgments on a DataWriter and a different thread writes new samples on the same DataWriter, the new samples must be acknowledged before unblocking the thread waiting on wait_for_acknowledgments.

If the **DDSDataWriter** (p. 1589) does not have **DDS_ReliabilityQosPolicy** (p. 1295) kind set to RELIABLE, this operation will succeed immediately.

Parameters

max_wait	*<<**in**>> (p. 433)* Specifies maximum time to wait for acknowledgements **DDS_Duration_t** (p. 1012) .

Exceptions

One	of the **Standard Return Codes** (p. 532), **DDS_RETCODE_NOT_ENABLED** (p. 533), **DDS_RETCODE_TIMEOUT** (p. 533)

9.250.2.21 is_sample_app_acknowledged()

```
virtual  DDS_ReturnCode_t DDSDataWriter::is_sample_app_acknowledged (
             DDS_Boolean & is_app_ack,
            const struct  DDS_SampleIdentity_t & identity )  [inline], [virtual]
```

This method can be used to see if a sample has been application acknowledged.

This method can be used to see if a sample has been application acknowledged by all the matching DataReaders that were alive when the sample was written.

Parameters

is_app_ack	*<<**out**>> (p. 433)* Cannot be NULL. This value will be set to **DDS_BOOLEAN_TRUE** (p. 513) when the sample has been acknowledged.
identity	*<<**in**>> (p. 433)* Cannot be NULL. Sample identity.

Exceptions

One	of the **Standard Return Codes** (p. 532)

Note

If the DataWriter does not match with any DataReader that enables application acknowledgments (that is, a DataReader with **DDS_ReliabilityQosPolicy::acknowledgment_kind** (p. 1298) set to a value other than **DDS_PROTOCOL_ACKNOWLEDGMENT_MODE** (p. 633)), the value of `is_app_ack` is set to **DDS_BOOLEAN_FALSE** (p. 513).

9.250.2.22 wait_for_asynchronous_publishing()

```
virtual  DDS_ReturnCode_t DDSDataWriter::wait_for_asynchronous_publishing (
            const  DDS_Duration_t & max_wait )  [inline], [virtual]
```

*<<**extension**>> (p. 432)* Blocks the calling thread until asynchronous sending is complete.

This operation blocks the calling thread (up to `max_wait`) until all data written by the asynchronous **DDSDataWriter** (p. 1589) is sent and acknowledged (if reliable) by all matched **DDSDataReader** (p. 1556) entities. A successful completion indicates that all the samples written have been sent and acknowledged where applicable; a time out indicates that `max_wait` elapsed before all the data was sent and/or acknowledged.

In other words, this guarantees that sending to best effort **DDSDataReader** (p. 1556) is complete in addition to what **DDSDataWriter::wait_for_acknowledgments** (p. 1602) provides.

If the **DDSDataWriter** (p. 1589) does not have **DDS_PublishModeQosPolicy** (p. 1279) kind set to **DDS_ASYNCHRONOUS_PUBLISH_MODE_QOS** (p. 629) the operation will complete immediately , with **DDS_RETCODE_OK** (p. 532).

Parameters

max_wait	*<<**in**>> (p. 433)* Specifies maximum time to wait for acknowledgements.

Exceptions

One	of the **Standard Return Codes** (p. 532), **DDS_RETCODE_NOT_ENABLED** (p. 533), **DDS_RETCODE_TIMEOUT** (p. 533)

9.250.2.23 set_qos()

```
virtual DDS_ReturnCode_t DDSDataWriter::set_qos (
            const DDS_DataWriterQos & qos ) [inline], [virtual]
```

Sets the writer QoS.

This operation modifies the QoS of the **DDSDataWriter** (p. 1589).

The **DDS_DataWriterQos::user_data** (p. 914), **DDS_DataWriterQos::deadline** (p. 912), **DDS_DataWriterQos::latency_budget** (p. 912), **DDS_DataWriterQos::ownership_strength** (p. 914), **DDS_DataWriterQos::transport_priority** (p. 913), **DDS_DataWriterQos::lifespan** (p. 914) and **DDS_DataWriterQos::writer_data_lifecycle** (p. 914) can be changed. The other policies are immutable.

Parameters

qos	*<<**in**>> (p. 433)* The **DDS_DataWriterQos** (p. 909) to be set to. Policies must be consistent. Immutable policies cannot be changed after **DDSDataWriter** (p. 1589) is enabled. The special value **DDS_DATAWRITER_QOS_DEFAULT** (p. 114) can be used to indicate that the QoS of the **DDSDataWriter** (p. 1589) should be changed to match the current default **DDS_DataWriterQos** (p. 909) set in the **DDSPublisher** (p. 1818).

Exceptions

One	of the **Standard Return Codes** (p. 532), **DDS_RETCODE_IMMUTABLE_POLICY** (p. 533) or **DDS_RETCODE_INCONSISTENT_POLICY** (p. 533)

See also

DDS_DataWriterQos (p. 909) for rules on consistency among QoS

set_qos (abstract) (p. 1732)

Operations Allowed in Listener Callbacks (p. 1796)

9.250.2.24 set_property()

```
virtual  DDS_ReturnCode_t DDSDataWriter::set_property (
             const char * property_name,
             const char * value,
             bool propagate )  [inline], [virtual]
```

Set the value for a property that applies to a DataWriter.

Warning

This method is not implemented in all APIs and it's intended only for testing purposes. You should use **DDSData↩ Writer::set_qos** (p. 1604) instead.

Parameters

property_name	*<<**in**>> (p. 433)*. Name of the property that you want to set.
value	*<<**in**>> (p. 433)*. New value for the property.
propagate	*<<**in**>> (p. 433)*. Indicates if the property will be propagated or not.

Exceptions

One	of the **Standard Return Codes** (p. 532)

See also

DDSDomainParticipant::set_property (p. 1676)

DDSDataReader::set_property (p. 1576)

DDSDataWriter::set_qos (p. 1604)

9.250.2.25 set_qos_with_profile()

```
virtual DDS_ReturnCode_t DDSDataWriter::set_qos_with_profile (
            const char * library_name,
            const char * profile_name ) [inline], [virtual]
```

<<***extension***>> *(p. 432)* Change the QoS of this writer using the input XML QoS profile.

This operation modifies the QoS of the **DDSDataWriter** (p. 1589).

The **DDS_DataWriterQos::user_data** (p. 914), **DDS_DataWriterQos::deadline** (p. 912), **DDS_DataWriterQos::latency_budget** (p. 912), **DDS_DataWriterQos::ownership_strength** (p. 914), **DDS_DataWriterQos::transport_priority** (p. 913), **DDS_DataWriterQos::lifespan** (p. 914) and **DDS_DataWriterQos::writer_data_lifecycle** (p. 914) can be changed. The other policies are immutable.

Parameters

library_name	<<***in***>> *(p. 433)* Library name containing the XML QoS profile. If library_name is null RTI Connext will use the default library (see **DDSPublisher::set_default_library** (p. 1823)).
profile_name	<<***in***>> *(p. 433)* XML QoS Profile name. If profile_name is null RTI Connext will use the default profile (see **DDSPublisher::set_default_profile** (p. 1824)).

Exceptions

One	of the **Standard Return Codes** (p. 532), **DDS_RETCODE_IMMUTABLE_POLICY** (p. 533) or **DDS_RETCODE_INCONSISTENT_POLICY** (p. 533)

See also

DDS_DataWriterQos (p. 909) for rules on consistency among QoS

Operations Allowed in Listener Callbacks (p. 1796)

9.250.2.26 get_qos()

```
virtual DDS_ReturnCode_t DDSDataWriter::get_qos (
             DDS_DataWriterQos & qos ) [inline], [virtual]
```

Gets the writer QoS.

This method may potentially allocate memory depending on the sequences contained in some QoS policies.

Parameters

qos	<<***inout***>> *(p. 433)* The **DDS_DataWriterQos** (p. 909) to be filled up.

Exceptions

One	of the **Standard Return Codes** (p. 532)

See also

get_qos (abstract) (p. 1733)

9.250.2.27 set_listener()

```
virtual DDS_ReturnCode_t DDSDataWriter::set_listener (
            DDSDataWriterListener * l,
            DDS_StatusMask mask = DDS_STATUS_MASK_ALL ) [inline], [virtual]
```

Sets the writer listener.

Parameters

l	<<***in***>> *(p. 433)* **DDSDataWriterListener** (p. 1612) to set to
mask	<<***in***>> *(p. 433)* **DDS_StatusMask** (p. 537) associated with the **DDSDataWriterListener** (p. 1612).

Exceptions

One	of the **Standard Return Codes** (p. 532)

See also

set_listener (abstract) (p. 1733)

9.250.2.28 get_listener()

```
virtual DDSDataWriterListener * DDSDataWriter::get_listener ( ) [inline], [virtual]
```

Get the writer listener.

Returns

DDSDataWriterListener (p. 1612) of the **DDSDataWriter** (p. 1589).

See also

get_listener (abstract) (p. 1734)

9.250.2.29 flush()

```
virtual DDS_ReturnCode_t DDSDataWriter::flush ( ) [inline], [virtual]
```

<<***extension***>> *(p. 432)* Flushes the batch in progress in the context of the calling thread.

After being flushed, the batch is available to be sent on the network.

If the **DDSDataWriter** (p. 1589) does not have **DDS_PublishModeQosPolicy** (p. 1279) kind set to **DDS_↩ ASYNCHRONOUS_PUBLISH_MODE_QOS** (p. 629), the batch will be sent on the network immediately (in the context of the calling thread).

If the **DDSDataWriter** (p. 1589) does have **DDS_PublishModeQosPolicy** (p. 1279) kind set to **DDS_ASYNCHRONOUS↩ _PUBLISH_MODE_QOS** (p. 629), the batch will be sent in the context of the asynchronous publishing thread.

This operation may block in the same conditions as **FooDataWriter::write** (p. 1951).

If this operation does block, the RELIABILITY max_blocking_time configures the maximum time the write operation may block (waiting for space to become available). If max_blocking_time elapses before the DDS_DataWriter is able to store the modification without exceeding the limits, the operation will fail with DDS_RETCODE_TIMEOUT.

MT Safety:

flush() (p. 1607) is only thread-safe with batching if **DDS_BatchQosPolicy::thread_safe_write** (p. 813) is TRUE.

Exceptions

One	of the **Standard Return Codes** (p. 532), **DDS_RETCODE_TIMEOUT** (p. 533), **DDS_RETCODE_OUT_OF_RESOURCES** (p. 533) or **DDS_RETCODE_NOT_ENABLED** (p. 533).

9.250.2.30 take_discovery_snapshot() [1/2]

```
virtual DDS_ReturnCode_t DDSDataWriter::take_discovery_snapshot ( ) [inline], [virtual]
```

Take a snapshot of the compatible and incompatible remote readers matched by a local writer.

The snapshot will be printed through the **NDDSConfigLogger** (p. 2093). A possible output may be the following:

```
Remote readers that match the local writer domain=0 name="writerTestName"

guid="0x0101D8D1,0x20B83C0D,0x4495246E:0x80000003"

topic="FooTopic" type="FooType"

------------------------------------------------------------------------

Compatible readers:

1. 0x0101542A,0x2C59B595,0xA1693BDF:0x80000004 name="reader1TestName"

kind="unkeyed user datareader"

unicastLocators="udpv4://192.168.1.170:7411"
```

```
liveliness="ALIVE"

Incompatible readers:

1. 0x0101542A,0x2C59B595,0xA1693BDF:0x80000104 name="reader2TestName"

kind="unkeyed user datareader"

unicastLocators="udpv4://192.168.1.170:7411"

reason="Inconsistent QoS"

------------------------------------------------------------------------
```

Exceptions

One	of the **Standard Return Codes** (p. 532).

9.250.2.31 take_discovery_snapshot() [2/2]

```
virtual DDS_ReturnCode_t DDSDataWriter::take_discovery_snapshot (
            const char * file_name ) [inline], [virtual]
```

Take a snapshot of the compatible and incompatible remote readers matched by a local writer.

The snapshot will be printed in the file specified by file_name. A possible output may be the following:

```
Remote readers that match the local writer domain=0 name="writerTestName"

guid="0x0101D8D1,0x20B83C0D,0x4495246E:0x80000003"

topic="FooTopic" type="FooType"

------------------------------------------------------------------------

Compatible readers:

1. 0x0101542A,0x2C59B595,0xA1693BDF:0x80000004 name="reader1TestName"

kind="unkeyed user datareader"

unicastLocators="udpv4://192.168.1.170:7411"

liveliness="ALIVE"

Incompatible readers:

1. 0x0101542A,0x2C59B595,0xA1693BDF:0x80000104 name="reader2TestName"

kind="unkeyed user datareader"

unicastLocators="udpv4://192.168.1.170:7411"

reason="Inconsistent QoS"

------------------------------------------------------------------------
```

Parameters

file_name	<<***in***>> *(p. 433)* Name of the file where snapshot should be printed.

Exceptions

One	of the **Standard Return Codes** (p. 532).

9.250.2.32 enable()

```
virtual DDS_ReturnCode_t DDSDataWriter::enable ( ) [inline], [virtual]
```

Enables the **DDSEntity** (p. 1731).

This operation enables the Entity. Entity objects can be created either enabled or disabled. This is controlled by the value of the **ENTITY_FACTORY** (p. 600) QoS policy on the corresponding factory for the **DDSEntity** (p. 1731).

By default, **ENTITY_FACTORY** (p. 600) is set so that it is not necessary to explicitly call **DDSEntity::enable** (p. 1734) on newly created entities.

The **DDSEntity::enable** (p. 1734) operation is idempotent. Calling enable on an already enabled Entity returns OK and has no effect.

If a **DDSEntity** (p. 1731) has not yet been enabled, the following kinds of operations may be invoked on it:

- set or get the QoS policies (including default QoS policies) and listener
- **DDSEntity::get_statuscondition** (p. 1735)
- 'factory' operations
- **DDSEntity::get_status_changes** (p. 1735) and other get status operations (although the status of a disabled entity never changes)
- 'lookup' operations

Other operations may explicitly state that they may be called on disabled entities; those that do not will return the error **DDS_RETCODE_NOT_ENABLED** (p. 533).

It is legal to delete an **DDSEntity** (p. 1731) that has not been enabled by calling the proper operation on its factory .

Entities created from a factory Entity that is disabled are created disabled, regardless of the setting of the **DDS_Entity↩FactoryQosPolicy** (p. 1145).

Calling enable on an Entity whose factory Entity is not enabled will fail and return **DDS_RETCODE_PRECONDITION↩_NOT_MET** (p. 532).

If **DDS_EntityFactoryQosPolicy::autoenable_created_entities** (p. 1147) is TRUE, the enable operation on a factory will automatically enable all entities created from that factory (for example, enabling a **DDSPublisher** (p. 1818) will enable all its contained **DDSDataWriter** (p. 1589) objects)

Listeners associated with an entity are not called until the entity is enabled.

Conditions associated with a disabled entity are "inactive," that is, they have a `trigger_value` == FALSE.

Exceptions

One	of the **Standard Return Codes** (p. 532), **Standard Return Codes** (p. 532) or **DDS_RETCODE_PRECONDITION_NOT_MET** (p. 532).

Implements **DDSEntity** (p. 1734).

9.250.2.33 get_statuscondition()

```
virtual DDSStatusCondition * DDSDataWriter::get_statuscondition ( ) [inline], [virtual]
```

Allows access to the **DDSStatusCondition** (p. 1846) associated with the **DDSEntity** (p. 1731).

The returned condition can then be added to a **DDSWaitSet** (p. 1897) so that the application can wait for specific status changes that affect the **DDSEntity** (p. 1731).

Returns

the status condition associated with this entity.

Implements **DDSEntity** (p. 1735).

9.250.2.34 get_status_changes()

```
virtual DDS_StatusMask DDSDataWriter::get_status_changes ( ) [inline], [virtual]
```

Retrieves the list of communication statuses in the **DDSEntity** (p. 1731) that are triggered.

That is, the list of statuses whose value has changed since the last time the application read the status using the get_∗_status() method.

When the entity is first created or if the entity is not enabled, all communication statuses are in the "untriggered" state so the list returned by the get_status_changes operation will be empty.

The list of statuses returned by the get_status_changes operation refers to the status that are triggered on the Entity itself and does not include statuses that apply to contained entities.

Returns

list of communication statuses in the **DDSEntity** (p. 1731) that are triggered.

See also

Status Kinds (p. 533)

Implements **DDSEntity** (p. 1735).

9.250.2.35 get_instance_handle()

```
virtual DDS_InstanceHandle_t DDSDataWriter::get_instance_handle ( ) [inline], [virtual]
```

Allows access to the **DDS_InstanceHandle_t** (p. 76) associated with the **DDSEntity** (p. 1731).

This operation returns the **DDS_InstanceHandle_t** (p. 76) that represents the **DDSEntity** (p. 1731).

Returns

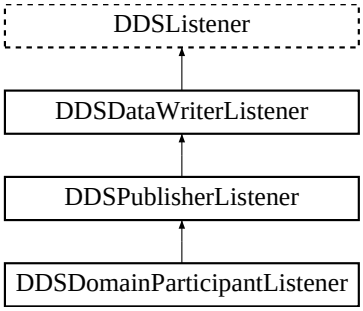
the instance handle associated with this entity.

Implements **DDSEntity** (p. 1736).

9.251 DDSDataWriterListener Class Reference

<<***interface***>> *(*p. *433)* **DDSListener** (p. 1794) for writer status.

Inheritance diagram for DDSDataWriterListener:



Public Member Functions

- virtual void **on_offered_deadline_missed** (**DDSDataWriter** ∗writer, const **DDS_OfferedDeadlineMissed↩ Status** &status)

  *Handles the **DDS_OFFERED_DEADLINE_MISSED_STATUS** (*p. *539) status.*
- virtual void **on_liveliness_lost** (**DDSDataWriter** ∗writer, const **DDS_LivelinessLostStatus** &status)

  *Handles the **DDS_LIVELINESS_LOST_STATUS** (*p. *542) status.*
- virtual void **on_offered_incompatible_qos** (**DDSDataWriter** ∗writer, const **DDS_OfferedIncompatibleQos↩ Status** &status)

  *Handles the **DDS_OFFERED_INCOMPATIBLE_QOS_STATUS** (*p. *540) status.*
- virtual void **on_publication_matched** (**DDSDataWriter** ∗writer, const **DDS_PublicationMatchedStatus** &status)

  *Handles the **DDS_PUBLICATION_MATCHED_STATUS** (*p. *543) status.*
- virtual void **on_reliable_writer_cache_changed** (**DDSDataWriter** ∗writer, const **DDS_ReliableWriterCache↩ ChangedStatus** &status)

  <<***extension***>> *(*p. *432) A change has occurred in the writer's cache of unacknowledged samples.*

- virtual void **on_reliable_reader_activity_changed** (**DDSDataWriter** ∗writer, const **DDS_ReliableReader↩ ActivityChangedStatus** &status)

  <<***extension***>> (p. *432) A matched reliable reader has become active or become inactive.*
- virtual void **on_sample_removed** (**DDSDataWriter** ∗writer, const **DDS_Cookie_t** &cookie)

  <<***extension***>> (p. *432) Called when a sample is removed from the DataWriter queue.*
- virtual void **on_instance_replaced** (**DDSDataWriter** ∗writer, const **DDS_InstanceHandle_t** &handle)

  <<***extension***>> (p. *432) Notifies when an instance is replaced in DataWriter queue.*
- virtual void **on_application_acknowledgment** (**DDSDataWriter** ∗writer, const **DDS_AcknowledgmentInfo** &info)

  <<***extension***>> (p. *432) Called when a sample is application-acknowledged*
- virtual void **on_service_request_accepted** (**DDSDataWriter** ∗writer, const **DDS_ServiceRequestAccepted↩ Status** &status)

  <<***extension***>> (p. *432) Called when a **DDS_ServiceRequest** (p. 1359) for the **DDSTopicQuery** (p. 1895) service is dispatched to this **DDSDataWriter** (p. 1589) for processing.*

9.251.1 Detailed Description

<<***interface***>> (p. *433)* **DDSListener** (p. 1794) for writer status.

Entity:

DDSDataWriter (p. 1589)

Status:

DDS_LIVELINESS_LOST_STATUS (p. 542), **DDS_LivelinessLostStatus** (p. 1179);
DDS_OFFERED_DEADLINE_MISSED_STATUS (p. 539), **DDS_OfferedDeadlineMissedStatus** (p. 1219);
DDS_OFFERED_INCOMPATIBLE_QOS_STATUS (p. 540), **DDS_OfferedIncompatibleQosStatus** (p. 1221);
DDS_PUBLICATION_MATCHED_STATUS (p. 543), **DDS_PublicationMatchedStatus** (p. 1273);
DDS_RELIABLE_READER_ACTIVITY_CHANGED_STATUS (p. 545), **DDS_ReliableReaderActivity↩ ChangedStatus** (p. 1300);
DDS_RELIABLE_WRITER_CACHE_CHANGED_STATUS (p. 545), **DDS_ReliableWriterCacheChanged↩ Status** (p. 1302);

See also

DDSListener (p. 1794)

Status Kinds (p. 533)

Operations Allowed in Listener Callbacks (p. 1796)

9.251.2 Member Function Documentation

9.251.2.1 on_offered_deadline_missed()

```
virtual void DDSDataWriterListener::on_offered_deadline_missed (
              DDSDataWriter * writer,
             const  DDS_OfferedDeadlineMissedStatus & status )  [virtual]
```

Handles the **DDS_OFFERED_DEADLINE_MISSED_STATUS** (p. 539) status.

This callback is called when the deadline that the **DDSDataWriter** (p. 1589) has committed through its **DEADLINE** (p. 583) qos policy was not respected for a specific instance. This callback is called for each deadline period elapsed during which the **DDSDataWriter** (p. 1589) failed to provide data for an instance.

Parameters

writer	<<***out***>> *(p. 433)* Locally created **DDSDataWriter** (p. 1589) that triggers the listener callback
status	<<***out***>> *(p. 433)* Current deadline missed status of locally created **DDSDataWriter** (p. 1589)

9.251.2.2 on_liveliness_lost()

```
virtual void DDSDataWriterListener::on_liveliness_lost (
              DDSDataWriter * writer,
             const  DDS_LivelinessLostStatus & status )  [virtual]
```

Handles the **DDS_LIVELINESS_LOST_STATUS** (p. 542) status.

This callback is called when the liveliness that the **DDSDataWriter** (p. 1589) has committed through its **LIVELINESS** (p. 607) qos policy was not respected; this **DDSDataReader** (p. 1556) entities will consider the **DDSDataWriter** (p. 1589) as no longer "alive/active". This callback will not be called when an already not alive **DDSDataWriter** (p. 1589) simply remains not alive for another liveliness period.

Parameters

writer	<<***out***>> *(p. 433)* Locally created **DDSDataWriter** (p. 1589) that triggers the listener callback
status	<<***out***>> *(p. 433)* Current liveliness lost status of locally created **DDSDataWriter** (p. 1589)

9.251.2.3 on_offered_incompatible_qos()

```
virtual void DDSDataWriterListener::on_offered_incompatible_qos (
              DDSDataWriter * writer,
             const  DDS_OfferedIncompatibleQosStatus & status )  [virtual]
```

Handles the **DDS_OFFERED_INCOMPATIBLE_QOS_STATUS** (p. 540) status.

This callback is called when the **DDS_DataWriterQos** (p. 909) of the **DDSDataWriter** (p. 1589) was incompatible with what was requested by a **DDSDataReader** (p. 1556). This callback is called when a **DDSDataWriter** (p. 1589) has discovered a **DDSDataReader** (p. 1556) for the same **DDSTopic** (p. 1885) and common partition, but with a requested QoS that is incompatible with that offered by the **DDSDataWriter** (p. 1589).

Parameters

writer	<<***out***>> *(p. 433)* Locally created **DDSDataWriter** (p. 1589) that triggers the listener callback
status	<<***out***>> *(p. 433)* Current incompatible qos status of locally created **DDSDataWriter** (p. 1589)

9.251.2.4 on_publication_matched()

```
virtual void DDSDataWriterListener::on_publication_matched (
              DDSDataWriter * writer,
             const  DDS_PublicationMatchedStatus & status )  [virtual]
```

Handles the **DDS_PUBLICATION_MATCHED_STATUS** (p. 543) status.

This callback is called when the **DDSDataWriter** (p. 1589) has found a **DDSDataReader** (p. 1556) that matches the **DDSTopic** (p. 1885), has a common partition and compatible QoS, or has ceased to be matched with a **DDSData↩ Reader** (p. 1556) that was previously considered to be matched.

Parameters

writer	<<***out***>> *(p. 433)* Locally created **DDSDataWriter** (p. 1589) that triggers the listener callback
status	<<***out***>> *(p. 433)* Current publication match status of locally created **DDSDataWriter** (p. 1589)

9.251.2.5 on_reliable_writer_cache_changed()

```
virtual void DDSDataWriterListener::on_reliable_writer_cache_changed (
              DDSDataWriter * writer,
             const  DDS_ReliableWriterCacheChangedStatus & status )  [virtual]
```

<<***extension***>> *(p. 432)* A change has occurred in the writer's cache of unacknowledged samples.

This listener callback is triggered when:

- The cache is empty (contains no unacknowledged samples).
- The cache is full (the number of unacknowledged samples has reached the value specified in **DDS_Resource↩ LimitsQosPolicy::max_samples** (p. 1311)).
- The number of unacknowledged samples has reached **DDS_RtpsReliableWriterProtocol_t::high_watermark** (p. 1322) or **DDS_RtpsReliableWriterProtocol_t::low_watermark** (p. 1321).

Parameters

writer	<<***out***>> *(p. 433)* Locally created **DDSDataWriter** (p. 1589) that triggers the listener callback
status	<<***out***>> *(p. 433)* Current reliable writer cache changed status of locally created **DDSDataWriter** (p. 1589)

9.251.2.6 on_reliable_reader_activity_changed()

```
virtual void DDSDataWriterListener::on_reliable_reader_activity_changed (
             DDSDataWriter * writer,
            const  DDS_ReliableReaderActivityChangedStatus & status )  [virtual]
```

<<***extension***>> *(p. 432)* A matched reliable reader has become active or become inactive.

Parameters

writer	<<***out***>> *(p. 433)* Locally created **DDSDataWriter** (p. 1589) that triggers the listener callback
status	<<***out***>> *(p. 433)* Current reliable reader activity changed status of locally created **DDSDataWriter** (p. 1589)

9.251.2.7 on_sample_removed()

```
virtual void DDSDataWriterListener::on_sample_removed (
             DDSDataWriter * writer,
            const  DDS_Cookie_t & cookie )  [virtual]
```

<<***extension***>> *(p. 432)* Called when a sample is removed from the DataWriter queue.

This callback is called only if the sample was written with a **DDS_Cookie_t** (p. 830) with **FooDataWriter::write_w_params** (p. 1956), or if this writer uses **Zero Copy** (p. 73) transfer over shared memory" or **FlatData Topic-Types** (p. 776) "FlatData language binding".

Parameters

writer	<<***out***>> *(p. 433)* Locally created **DDSDataWriter** (p. 1589) that triggers the listener callback
cookie	<<***out***>> *(p. 433)* • If this sample was written with **FooDataWriter::write_w_params** (p. 1956), this field contains a copy of the cookie set in **DDS_WriteParams_t** (p. 1519). • If this writer uses **Zero Copy transfer over shared memory** (p. 73) or **FlatData language binding** (p. 776), this field contains the absolute address of the sample that is removed. The address of the sample can be obtained by using **DDS_Cookie_t::to_pointer** (p. 671)

See also

FooDataWriter::get_loan (p. 1963)

9.251.2.8 on_instance_replaced()

```
virtual void DDSDataWriterListener::on_instance_replaced (
              DDSDataWriter * writer,
             const  DDS_InstanceHandle_t & handle )  [virtual]
```

<<***extension***>> *(p. 432)* Notifies when an instance is replaced in DataWriter queue.

This callback is called when an instance is replaced by the **DDSDataWriter** (p. 1589) due to instance resource limits being reached. This callback returns to the user the handle of the replaced instance, which can be used to get the key of the replaced instance using the **FooDataWriter::get_key_value** (p. 1959) API.

Because this callback can be called within the context of an in-progress write, dispose, or unregister call, most APIs on the DataWriter must not be used. The only DataWriter APIs that are safe to call within this callback are:

- **FooDataWriter::get_key_value** (p. 1959)
- **FooDataWriter::narrow** (p. 1946)
- **FooDataWriter::create_data** (p. 1962)
- **FooDataWriter::delete_data** (p. 1962)
- **DDSDataWriter::get_matched_subscriptions** (p. 1599)
- **DDSDataWriter::is_matched_subscription_active** (p. 1596)
- **DDSDataWriter::get_matched_subscription_participant_data** (p. 1601)
- **DDSDataWriter::get_topic** (p. 1601)
- **DDSDataWriter::get_publisher** (p. 1602)
- **DDSDataWriter::is_sample_app_acknowledged** (p. 1603)

Parameters

writer	<<***out***>> *(p. 433)* Locally created **DDSDataWriter** (p. 1589) that triggers the listener callback
handle	<<***out***>> *(p. 433)* Handle of the replaced instance

9.251.2.9 on_application_acknowledgment()

```
virtual void DDSDataWriterListener::on_application_acknowledgment (
```

```
            DDSDataWriter * writer,
          const  DDS_AcknowledgmentInfo & info )  [virtual]
```

<<***extension***>> *(p. 432)* Called when a sample is application-acknowledged

Applicable only when **DDS_ReliabilityQosPolicy::acknowledgment_kind** (p. 1298) = **DDS_APPLICATION_AUTO_ACKNOWLEDGMENT_MODE** (p. 633) or **DDS_APPLICATION_EXPLICIT_ACKNOWLEDGMENT_MODE** (p. 634)

Called when a sample is application-level acknowledged. Provides identities of the sample and the acknowledging **DDSDataReader** (p. 1556). Also provides user-specified response data sent from the **DDSDataReader** (p. 1556) by the acknowledgment message.

Parameters

writer	<<***out***>> *(p. 433)* Locally created **DDSDataWriter** (p. 1589) that triggers the listener callback
info	<<***out***>> *(p. 433)* **DDS_AcknowledgmentInfo** (p. 794) of the acknowledged sample

9.251.2.10 on_service_request_accepted()

```
virtual void DDSDataWriterListener::on_service_request_accepted (
            DDSDataWriter * writer,
          const  DDS_ServiceRequestAcceptedStatus & status )  [virtual]
```

<<***extension***>> *(p. 432)* Called when a **DDS_ServiceRequest** (p. 1359) for the **DDSTopicQuery** (p. 1895) service is dispatched to this **DDSDataWriter** (p. 1589) for processing.

Parameters

writer	<<***out***>> *(p. 433)* Locally created **DDSDataWriter** (p. 1589) that triggers the listener callback
status	<<***out***>> *(p. 433)* Current service request accepted status of locally created **DDSDataWriter** (p. 1589)

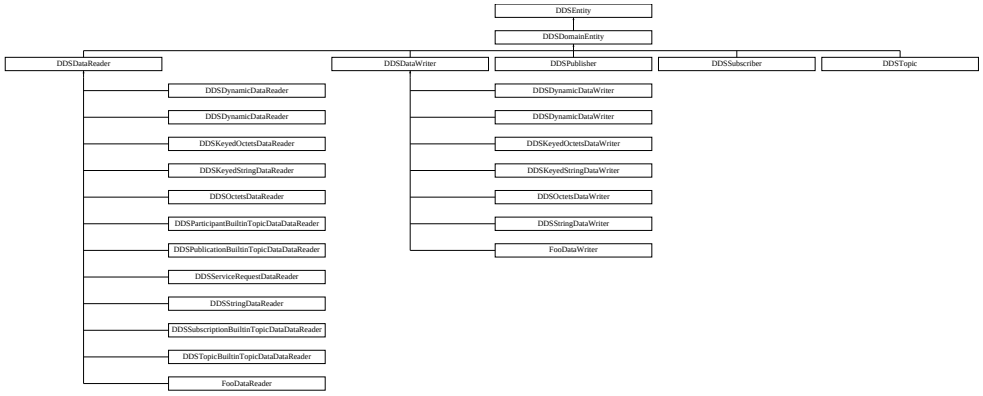
See also

Topic Queries (p. 155)

9.252 DDSDomainEntity Class Reference

<<***interface***>> *(p. 433)* Abstract base class for all DDS entities except for the **DDSDomainParticipant** (p. 1619).

Inheritance diagram for DDSDomainEntity:



Additional Inherited Members

9.252.1 Detailed Description

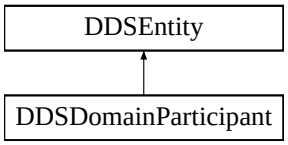
<<***interface***>> *(*p. *433)* Abstract base class for all DDS entities except for the **DDSDomainParticipant** (p. 1619).

Its sole purpose is to *conceptually* express that **DDSDomainParticipant** (p. 1619) is a special kind of **DDSEntity** (p. 1731) that acts as a container of all other **DDSEntity** (p. 1731) but itself cannot contain other **DDSDomainParticipant** (p. 1619).

9.253 DDSDomainParticipant Class Reference

<<***interface***>> *(*p. *433)* Container for all **DDSDomainEntity** (p. 1618) objects.

Inheritance diagram for DDSDomainParticipant:



Public Member Functions

- virtual **DDS_ReturnCode_t get_default_datawriter_qos** (**DDS_DataWriterQos** &qos)=0

  <<***extension***>> *(*p. *432) Copy the default **DDS_DataWriterQos** (*p. *909) values into the provided **DDS_DataWriterQos** (*p. *909) instance.*
- virtual **DDS_ReturnCode_t set_default_datawriter_qos** (const **DDS_DataWriterQos** &qos)=0

  <<***extension***>> *(*p. *432) Set the default DataWriterQos values for this DomainParticipant.*
- virtual **DDS_ReturnCode_t set_default_datawriter_qos_with_profile** (const char ∗library_name, const char ∗profile_name)=0

*<<**extension**>> (p. 432) Set the default **DDS_DataWriterQos** (p. 909) values for this domain participant based on the input XML QoS profile.*

- virtual **DDS_ReturnCode_t get_default_datareader_qos** (**DDS_DataReaderQos** &qos)=0

  *<<**extension**>> (p. 432) Copy the default **DDS_DataReaderQos** (p. 858) values into the provided **DDS_DataReader_Qos** (p. 858) instance.*
- virtual **DDS_ReturnCode_t set_default_datareader_qos** (const **DDS_DataReaderQos** &qos)=0

  *<<**extension**>> (p. 432) Set the default **DDS_DataReaderQos** (p. 858) values for this domain participant.*
- virtual **DDS_ReturnCode_t set_default_datareader_qos_with_profile** (const char *library_name, const char *profile_name)=0

  *<<**extension**>> (p. 432) Set the default **DDS_DataReaderQos** (p. 858) values for this DomainParticipant based on the input XML QoS profile.*
- virtual **DDS_ReturnCode_t get_default_flowcontroller_property** (**DDS_FlowControllerProperty_t** &prop)=0

  *<<**extension**>> (p. 432) Copies the default **DDS_FlowControllerProperty_t** (p. 1156) values for this domain participant into the given **DDS_FlowControllerProperty_t** (p. 1156) instance.*
- virtual **DDS_ReturnCode_t set_default_flowcontroller_property** (const **DDS_FlowControllerProperty_t** &prop)=0

  *<<**extension**>> (p. 432) Set the default **DDS_FlowControllerProperty_t** (p. 1156) values for this domain participant.*
- virtual **DDS_ReturnCode_t register_contentfilter** (const char *filter_name, const **DDSContentFilter** *contentfilter)=0

  *<<**extension**>> (p. 432) Register a content filter which can be used to create a **DDSContentFilteredTopic** (p. 1551).*
- virtual **DDSContentFilter** * **lookup_contentfilter** (const char *filter_name)=0

  *<<**extension**>> (p. 432) Lookup a content filter previously registered with **DDSDomainParticipant::register_contentfilter** (p. 1632).*
- virtual **DDS_ReturnCode_t unregister_contentfilter** (const char *filter_name)=0

  *<<**extension**>> (p. 432) Unregister a content filter previously registered with **DDSDomainParticipant::register_contentfilter** (p. 1632).*
- virtual const char * **get_default_library** ()=0

  *<<**extension**>> (p. 432) Gets the default XML library associated with a **DDSDomainParticipant** (p. 1619).*
- virtual const char * **get_default_profile** ()=0

  *<<**extension**>> (p. 432) Gets the default XML profile associated with a **DDSDomainParticipant** (p. 1619).*
- virtual const char * **get_default_profile_library** ()=0

  *<<**extension**>> (p. 432) Gets the library where the default XML QoS profile is contained for a **DDSDomainParticipant** (p. 1619).*
- virtual **DDS_ReturnCode_t set_default_library** (const char *library_name)=0

  *<<**extension**>> (p. 432) Sets the default XML library for a **DDSDomainParticipant** (p. 1619).*
- virtual **DDS_ReturnCode_t set_default_profile** (const char *library_name, const char *profile_name)=0

  *<<**extension**>> (p. 432) Sets the default XML profile for a **DDSDomainParticipant** (p. 1619).*
- virtual **DDS_ReturnCode_t get_default_topic_qos** (**DDS_TopicQos** &qos)=0

  *Copies the default **DDS_TopicQos** (p. 1397) values for this domain participant into the given **DDS_TopicQos** (p. 1397) instance.*
- virtual **DDS_ReturnCode_t set_default_topic_qos** (const **DDS_TopicQos** &qos)=0

  *Set the default **DDS_TopicQos** (p. 1397) values for this domain participant.*
- virtual **DDS_ReturnCode_t set_default_topic_qos_with_profile** (const char *library_name, const char *profile_name)=0

  *<<**extension**>> (p. 432) Set the default **DDS_TopicQos** (p. 1397) values for this domain participant based on the input XML QoS profile.*
- virtual **DDS_ReturnCode_t get_default_publisher_qos** (**DDS_PublisherQos** &qos)=0

  *Copy the default **DDS_PublisherQos** (p. 1276) values into the provided **DDS_PublisherQos** (p. 1276) instance.*
- virtual **DDS_ReturnCode_t set_default_publisher_qos** (const **DDS_PublisherQos** &qos)=0

*Set the default **DDS_PublisherQos** (p. 1276) values for this DomainParticipant.*

- virtual **DDS_ReturnCode_t set_default_publisher_qos_with_profile** (const char *library_name, const char *profile_name)=0

  *<<**extension**>> (p. 432) Set the default **DDS_PublisherQos** (p. 1276) values for this DomainParticipant based on the input XML QoS profile.*

- virtual **DDS_ReturnCode_t get_default_subscriber_qos** (**DDS_SubscriberQos** &qos)=0

  *Copy the default **DDS_SubscriberQos** (p. 1367) values into the provided **DDS_SubscriberQos** (p. 1367) instance.*

- virtual **DDS_ReturnCode_t set_default_subscriber_qos** (const **DDS_SubscriberQos** &qos)=0

  *Set the default **DDS_SubscriberQos** (p. 1367) values for this DomainParticipant.*

- virtual **DDS_ReturnCode_t set_default_subscriber_qos_with_profile** (const char *library_name, const char *profile_name)=0

  *<<**extension**>> (p. 432) Set the default **DDS_SubscriberQos** (p. 1367) values for this DomainParticipant based on the input XML QoS profile.*

- virtual **DDSPublisher** * **create_publisher** (const **DDS_PublisherQos** &qos, **DDSPublisherListener** *listener, **DDS_StatusMask** mask)=0

  *Creates a **DDSPublisher** (p. 1818) with the desired QoS policies and attaches to it the specified **DDSPublisherListener** (p. 1839).*

- virtual **DDSPublisher** * **create_publisher_with_profile** (const char *library_name, const char *profile_name, **DDSPublisherListener** *listener, **DDS_StatusMask** mask)=0

  *<<**extension**>> (p. 432) Creates a new **DDSPublisher** (p. 1818) object using the **DDS_PublisherQos** (p. 1276) associated with the input XML QoS profile.*

- virtual **DDS_ReturnCode_t delete_publisher** (**DDSPublisher** *p)=0

  *Deletes an existing **DDSPublisher** (p. 1818).*

- virtual **DDSSubscriber** * **create_subscriber** (const **DDS_SubscriberQos** &qos, **DDSSubscriberListener** *listener, **DDS_StatusMask** mask)=0

  *Creates a **DDSSubscriber** (p. 1860) with the desired QoS policies and attaches to it the specified **DDSSubscriberListener** (p. 1881).*

- virtual **DDSSubscriber** * **create_subscriber_with_profile** (const char *library_name, const char *profile_name, **DDSSubscriberListener** *listener, **DDS_StatusMask** mask)=0

  *<<**extension**>> (p. 432) Creates a new **DDSSubscriber** (p. 1860) object using the **DDS_PublisherQos** (p. 1276) associated with the input XML QoS profile.*

- virtual **DDS_ReturnCode_t delete_subscriber** (**DDSSubscriber** *s)=0

  *Deletes an existing **DDSSubscriber** (p. 1860).*

- virtual **DDS_ReturnCode_t get_publishers** (**DDSPublisherSeq** &publishers)=0

  *<<**extension**>> (p. 432) Allows the application to access all the publishers the participant has.*

- virtual **DDS_ReturnCode_t get_subscribers** (**DDSSubscriberSeq** &subscribers)=0

  *<<**extension**>> (p. 432) Allows the application to access all the subscribers the participant has.*

- virtual **DDSTopic** * **create_topic** (const char *topic_name, const char *type_name, const **DDS_TopicQos** &qos, **DDSTopicListener** *listener, **DDS_StatusMask** mask)=0

  *Creates a **DDSTopic** (p. 1885) with the desired QoS policies and attaches to it the specified **DDSTopicListener** (p. 1894).*

- virtual **DDSTopic** * **create_topic_with_profile** (const char *topic_name, const char *type_name, const char *library_name, const char *profile_name, **DDSTopicListener** *listener, **DDS_StatusMask** mask)=0

  *<<**extension**>> (p. 432) Creates a new **DDSTopic** (p. 1885) object using the **DDS_PublisherQos** (p. 1276) associated with the input XML QoS profile.*

- virtual **DDS_ReturnCode_t delete_topic** (**DDSTopic** *topic)=0

  *Deletes a **DDSTopic** (p. 1885).*

- virtual **DDSContentFilteredTopic** * **create_contentfilteredtopic** (const char *name, **DDSTopic** *related_topic, const char *filter_expression, const **DDS_StringSeq** &expression_parameters)=0

  *Creates a **DDSContentFilteredTopic** (p. 1551), that can be used to do content-based subscriptions.*

- virtual **DDSContentFilteredTopic** * **create_contentfilteredtopic_with_filter** (const char *name, **DDSTopic** *related_topic, const char *filter_expression, const **DDS_StringSeq** &expression_parameters, const char *filter_name= **DDS_SQLFILTER_NAME**)=0

  *<<**extension**>> (p. 432) Creates a **DDSContentFilteredTopic** (p. 1551) using the specified filter to do content-based subscriptions.*
- virtual **DDS_ReturnCode_t** **delete_contentfilteredtopic** (**DDSContentFilteredTopic** *a_contentfilteredtopic)=0

  *Deletes a **DDSContentFilteredTopic** (p. 1551).*
- virtual **DDSMultiTopic** * **create_multitopic** (const char *name, const char *type_name, const char *subscription_expression, const **DDS_StringSeq** &expression_parameters)=0

  ***[Not supported (optional)]** Creates a MultiTopic that can be used to subscribe to multiple topics and combine/filter the received data into a resulting type.*
- virtual **DDS_ReturnCode_t** **delete_multitopic** (**DDSMultiTopic** *a_multitopic)=0

  ***[Not supported (optional)]** Deletes a **DDSMultiTopic** (p. 1797).*
- virtual **DDSTopic** * **find_topic** (const char *topic_name, const **DDS_Duration_t** &timeout)=0

  *Finds an existing (or ready to exist) **DDSTopic** (p. 1885), based on its name.*
- virtual **DDSTopicDescription** * **lookup_topicdescription** (const char *topic_name)=0

  *Looks up an existing, locally created **DDSTopicDescription** (p. 1892), based on its name.*
- virtual **DDSFlowController** * **create_flowcontroller** (const char *name, const **DDS_FlowController**↩ **Property_t** &prop)=0

  *<<**extension**>> (p. 432) Creates a **DDSFlowController** (p. 1736) with the desired property.*
- virtual **DDS_ReturnCode_t** **delete_flowcontroller** (**DDSFlowController** *fc)=0

  *<<**extension**>> (p. 432) Deletes an existing **DDSFlowController** (p. 1736).*
- virtual **DDSFlowController** * **lookup_flowcontroller** (const char *name)=0

  *<<**extension**>> (p. 432) Looks up an existing locally-created **DDSFlowController** (p. 1736), based on its name.*
- virtual **DDSSubscriber** * **get_builtin_subscriber** ()=0

  *Accesses the **built-in DDSSubscriber** (p. 1860).*
- virtual **DDS_ReturnCode_t** **ignore_participant** (const **DDS_InstanceHandle_t** &handle)=0

  *Instructs RTI Connext to locally ignore a remote **DDSDomainParticipant** (p. 1619).*
- virtual **DDS_ReturnCode_t** **banish_ignored_participants** ()=0

  *<<**extension**>> (p. 432) Prevents ignored remote DomainParticipants from receiving traffic from the local **DDSDomainParticipant** (p. 1619).*
- virtual **DDS_ReturnCode_t** **ignore_topic** (const **DDS_InstanceHandle_t** &handle)=0

  *Instructs RTI Connext to locally ignore a **DDSTopic** (p. 1885).*
- virtual **DDS_ReturnCode_t** **ignore_publication** (const **DDS_InstanceHandle_t** &handle)=0

  Instructs RTI Connext to locally ignore a publication.
- virtual **DDS_ReturnCode_t** **ignore_subscription** (const **DDS_InstanceHandle_t** &handle)=0

  Instructs RTI Connext to locally ignore a subscription.
- virtual **DDS_DomainId_t** **get_domain_id** ()=0

  Get the unique domain identifier.
- virtual **DDS_ReturnCode_t** **get_current_time** (**DDS_Time_t** &current_time)=0

  Returns the current value of the time.
- virtual **DDS_ReturnCode_t** **register_durable_subscription** (const **DDS_EndpointGroup_t** &group, const char *topic_name)=0

  *<<**extension**>> (p. 432) Registers a Durable Subscription on the specified **DDSTopic** (p. 1885) on all Persistence Services.*
- virtual **DDS_ReturnCode_t** **delete_durable_subscription** (const **DDS_EndpointGroup_t** &group)=0

  *<<**extension**>> (p. 432) Deletes an existing Durable Subscription on all Persistence Services.*
- virtual **DDS_ReturnCode_t** **assert_liveliness** ()=0

*Manually asserts the liveliness of this **DDSDomainParticipant** (p. 1619).*

- virtual **DDS_ReturnCode_t resume_endpoint_discovery** (const **DDS_InstanceHandle_t** &remote_participant_handle)=0

  *<<**extension**>> (p. 432) Initiates endpoint discovery with the specified remote **DDSDomainParticipant** (p. 1619).*

- virtual **DDS_ReturnCode_t delete_contained_entities** ()=0

  *Delete all the entities that were created by means of the "create" operations on the **DDSDomainParticipant** (p. 1619).*

- virtual **DDS_ReturnCode_t get_discovered_participants** (**DDS_InstanceHandleSeq** &participant_handles)=0

  *Returns a list of discovered **DDSDomainParticipant** (p. 1619) entities.*

- virtual **DDS_ReturnCode_t get_discovered_participants_from_subject_name** (**DDS_InstanceHandleSeq** &participant_handles, const char *subject_name)=0

  *<<**extension**>> (p. 432) Returns a list of discovered **DDSDomainParticipant** (p. 1619) entities that have the given **DDS_EntityNameQosPolicy::name** (p. 1148).*

- virtual **DDS_ReturnCode_t get_discovered_participant_data** (struct **DDS_ParticipantBuiltinTopicData** &participant_data, const **DDS_InstanceHandle_t** &participant_handle)=0

  *Returns **DDS_ParticipantBuiltinTopicData** (p. 1229) for the specified **DDSDomainParticipant** (p. 1619).*

- virtual **DDS_ReturnCode_t get_discovered_participant_subject_name** (char *subject_name, **DDS_UnsignedLong** &subject_name_size, const **DDS_InstanceHandle_t** &participant_handle)=0

  *<<**extension**>> (p. 432) Returns **DDS_EntityNameQosPolicy::name** (p. 1148) for the specified **DDSDomainParticipant** (p. 1619).*

- virtual **DDS_ReturnCode_t get_discovered_topics** (**DDS_InstanceHandleSeq** &topic_handles)=0

  *Returns list of discovered **DDSTopic** (p. 1885) objects.*

- virtual **DDS_ReturnCode_t get_discovered_topic_data** (struct **DDS_TopicBuiltinTopicData** &topic_data, const **DDS_InstanceHandle_t** &topic_handle)=0

  *Returns **DDS_TopicBuiltinTopicData** (p. 1391) for the specified **DDSTopic** (p. 1885).*

- virtual **DDS_Boolean contains_entity** (const **DDS_InstanceHandle_t** &a_handle)=0

  *Completes successfully with **DDS_BOOLEAN_TRUE** (p. 513) if the referenced **DDSEntity** (p. 1731) is contained by the **DDSDomainParticipant** (p. 1619).*

- virtual const **DDS_TypeCode** * **get_typecode** (const char *type_name)=0

  *<<**extension**>> (p. 432) Retrieves the **DDS_TypeCode** (p. 1428) of a type registered with the **DDSDomainParticipant** (p. 1619) based on the registration name.*

- virtual **DDS_ReturnCode_t get_participant_protocol_status** (struct **DDS_DomainParticipantProtocolStatus** &status)=0

  *<<**extension**>> (p. 432) Get the domain participant protocol status for this participant.*

- virtual **DDS_ReturnCode_t set_property** (const char *property_name, const char *value, bool propagate)=0

  Set the value for a property that applies to a DomainParticipant.

- virtual **DDS_ReturnCode_t set_qos** (const **DDS_DomainParticipantQos** &qos)=0

  Change the QoS of this DomainParticipant.

- virtual **DDS_ReturnCode_t set_qos_with_profile** (const char *library_name, const char *profile_name)=0

  *<<**extension**>> (p. 432) Change the QoS of this domain participant using the input XML QoS profile.*

- virtual **DDS_ReturnCode_t get_qos** (**DDS_DomainParticipantQos** &qos)=0

  Get the participant QoS.

- virtual **DDS_ReturnCode_t add_peer** (const char *peer_desc_string)=0

  *<<**extension**>> (p. 432) Attempt to contact one or more additional peer participants.*

- virtual **DDS_ReturnCode_t remove_peer** (const char *peer_desc_string)=0

  *<<**extension**>> (p. 432) Remove one or more peer participants from the list of peers with which this **DDSDomainParticipant** (p. 1619) will try to communicate.*

- virtual **DDS_ReturnCode_t get_dns_tracker_polling_period** (**DDS_Duration_t** &polling_period)=0

  *<<**extension**>> (p. 432) Retrieves the frequency used by the DNS tracker thread to query the DNS service.*

- virtual **DDS_ReturnCode_t set_dns_tracker_polling_period** (const **DDS_Duration_t** &polling_period)=0

  <<***extension***>> (p. *432) Configures the frequency in which the DNS tracker queries the DNS service.*

- virtual **DDS_ReturnCode_t set_listener** (**DDSDomainParticipantListener** *l, **DDS_StatusMask** mask= **DDS_STATUS_MASK_ALL**)=0

  Sets the participant listener.

- virtual **DDSDomainParticipantListener** * **get_listener** ()=0

  Get the participant listener.

- virtual **DDSPublisher** * **get_implicit_publisher** ()=0

  <<***extension***>> (p. *432) Returns the implicit **DDSPublisher** (p. 1818). If an implicit Publisher does not already exist, this creates one.*

- virtual **DDSSubscriber** * **get_implicit_subscriber** ()=0

  <<***extension***>> (p. *432) Returns the implicit **DDSSubscriber** (p. 1860). If an implicit Subscriber does not already exist, this creates one.*

- virtual **DDSDataWriter** * **create_datawriter** (**DDSTopic** *topic, const **DDS_DataWriterQos** &qos, **DDSDataWriterListener** *listener, **DDS_StatusMask** mask)=0

  <<***extension***>> (p. *432) Creates a **DDSDataWriter** (p. 1589) that will be attached and belong to the implicit **DDSPublisher** (p. 1818).*

- virtual **DDSDataWriter** * **create_datawriter_with_profile** (**DDSTopic** *topic, const char *library_name, const char *profile_name, **DDSDataWriterListener** *listener, **DDS_StatusMask** mask)=0

  <<***extension***>> (p. *432) Creates a **DDSDataWriter** (p. 1589) using a XML QoS profile that will be attached and belong to the implicit **DDSPublisher** (p. 1818).*

- virtual **DDS_ReturnCode_t delete_datawriter** (**DDSDataWriter** *a_datawriter)=0

  <<***extension***>> (p. *432) Deletes a **DDSDataWriter** (p. 1589) that belongs to the implicit **DDSPublisher** (p. 1818).*

- virtual **DDSDataReader** * **create_datareader** (**DDSTopicDescription** *topic, const **DDS_DataReaderQos** &qos, **DDSDataReaderListener** *listener, **DDS_StatusMask** mask)=0

  <<***extension***>> (p. *432) Creates a **DDSDataReader** (p. 1556) that will be attached and belong to the implicit **DDSSubscriber** (p. 1860).*

- virtual **DDSDataReader** * **create_datareader_with_profile** (**DDSTopicDescription** *topic, const char *library_name, const char *profile_name, **DDSDataReaderListener** *listener, **DDS_StatusMask** mask)=0

  <<***extension***>> (p. *432) Creates a **DDSDataReader** (p. 1556) using a XML QoS profile that will be attached and belong to the implicit **DDSSubscriber** (p. 1860).*

- virtual **DDS_ReturnCode_t delete_datareader** (**DDSDataReader** *a_datareader)=0

  <<***extension***>> (p. *432) Deletes a **DDSDataReader** (p. 1556) that belongs to the implicit **DDSSubscriber** (p. 1860).*

- virtual **DDSPublisher** * **lookup_publisher_by_name** (const char *publisher_name)=0

  <<***extension***>> (p. *432) Looks up a **DDSPublisher** (p. 1818) by its entity name within this **DDSDomainParticipant** (p. 1619).*

- virtual **DDSSubscriber** * **lookup_subscriber_by_name** (const char *subscriber_name)=0

  <<***extension***>> (p. *432) Retrieves a **DDSSubscriber** (p. 1860) by its entity name within this **DDSDomainParticipant** (p. 1619).*

- virtual **DDSDataWriter** * **lookup_datawriter_by_name** (const char *datawriter_full_name)=0

  <<***extension***>> (p. *432) Looks up a **DDSDataWriter** (p. 1589) by its entity name within this **DDSDomainParticipant** (p. 1619).*

- virtual **DDSDataReader** * **lookup_datareader_by_name** (const char *datareader_full_name)=0

  <<***extension***>> (p. *432) Retrieves up a **DDSDataReader** (p. 1556) by its entity name in this **DDSDomainParticipant** (p. 1619).*

- virtual **DDS_ReturnCode_t take_discovery_snapshot** ()=0

  Take a snapshot of the remote participants discovered by a local one.

- virtual **DDS_ReturnCode_t take_discovery_snapshot** (const char *file_name)=0

  Take a snapshot of the remote participants discovered by a local one.

9.253.1 Detailed Description

<<***interface***>> *(*p. *433)* Container for all **DDSDomainEntity** (p. 1618) objects.

The DomainParticipant object plays several roles:

- It acts as a container for all other **DDSEntity** (p. 1731) objects.
- It acts as a *factory* for the **DDSPublisher** (p. 1818), **DDSSubscriber** (p. 1860), **DDSTopic** (p. 1885) and **DDSMultiTopic** (p. 1797) **DDSEntity** (p. 1731) objects.
- It represents the participation of the application on a communication plane that isolates applications running on the same set of physical computers from each other. A domain establishes a virtual network linking all applications that share the same `domainId` and isolating them from applications running on different domains. In this way, several independent distributed applications can coexist in the same physical network without interfering, or even being aware of each other.
- It provides administration services in the domain, offering operations that allow the application to ignore locally any information about a given participant (**DDSDomainParticipant::ignore_participant** (p. 1661)), publication (**DDSDomainParticipant::ignore_publication** (p. 1664)), subscription (**DDSDomainParticipant::ignore_subscription()** (p. 1664)) or topic (**DDSDomainParticipant::ignore_topic()** (p. 1663)).

The following operations may be called even if the **DDSDomainParticipant** (p. 1619) is not enabled. Operations NOT in this list will fail with **DDS_RETCODE_NOT_ENABLED** (p. 533) \ if called on a disabled DomainParticipant.

- **DDSEntity::enable** (p. 1734),
- **DDSDomainParticipant::set_qos** (p. 1677), **DDSDomainParticipant::set_qos_with_profile** (p. 1677), **DDSDomainParticipant::get_qos** (p. 1678),
- **DDSDomainParticipant::set_listener** (p. 1682), **DDSDomainParticipant::get_listener** (p. 1682),
- Factory operations: **DDSDomainParticipant::create_flowcontroller** (p. 1659), **DDSDomainParticipant::create_topic** (p. 1650), **DDSDomainParticipant::create_topic_with_profile** (p. 1651), **DDSDomainParticipant::create_publisher** (p. 1644), **DDSDomainParticipant::create_publisher_with_profile** (p. 1645), **DDSDomainParticipant::create_subscriber** (p. 1647), **DDSDomainParticipant::create_subscriber_with_profile** (p. 1648), **DDSDomainParticipant::delete_flowcontroller** (p. 1660), **DDSDomainParticipant::delete_topic** (p. 1653), **DDSDomainParticipant::delete_publisher** (p. 1646), **DDSDomainParticipant::delete_subscriber** (p. 1648), **DDSDomainParticipant::set_default_topic_qos** (p. 1637), **DDSDomainParticipant::set_default_topic_qos_with_profile** (p. 1638), **DDSDomainParticipant::get_default_topic_qos** (p. 1636), **DDSDomainParticipant::set_default_publisher_qos** (p. 1640), **DDSDomainParticipant::set_default_publisher_qos_with_profile** (p. 1640), **DDSDomainParticipant::get_default_publisher_qos** (p. 1639), **DDSDomainParticipant::set_default_subscriber_qos** (p. 1642), **DDSDomainParticipant::set_default_subscriber_qos_with_profile** (p. 1643), **DDSDomainParticipant::get_default_subscriber_qos** (p. 1641), **DDSDomainParticipant::delete_contained_entities** (p. 1669), **DDSDomainParticipant::set_default_datareader_qos** (p. 1629), **DDSDomainParticipant::set_default_datareader_qos_with_profile** (p. 1630), **DDSDomainParticipant::get_default_datareader_qos** (p. 1628), **DDSDomainParticipant::set_default_datawriter_qos** (p. 1627), **DDSDomainParticipant::set_default_datawriter_qos_with_profile** (p. 1627), **DDSDomainParticipant::get_default_datawriter_qos** (p. 1626), **DDSDomainParticipant::set_default_library** (p. 1635), **DDSDomainParticipant::set_default_profile** (p. 1636) **DDSDomainParticipant::set_default_flowcontroller_property** (p. 1631), **DDSDomainParticipant::get_default_flowcontroller_property** (p. 1630), ;

- Operations for looking up topics: **DDSDomainParticipant::lookup_topicdescription** (p. 1658);

- Operations that access status: **DDSEntity::get_statuscondition** (p. 1735), **DDSEntity::get_status_changes** (p. 1735).

QoS:

DDS_DomainParticipantQos (p. 971)

Status:

Status Kinds (p. 533)

Listener:

DDSDomainParticipantListener (p. 1722)

See also

Operations Allowed in Listener Callbacks (p. 1796)

Examples

HelloWorld_publisher.cxx, and **HelloWorld_subscriber.cxx**.

9.253.2 Member Function Documentation

9.253.2.1 get_default_datawriter_qos()

```
virtual DDS_ReturnCode_t DDSDomainParticipant::get_default_datawriter_qos (
            DDS_DataWriterQos & qos ) [pure virtual]
```

<<***extension***>> *(p. 432)* Copy the default **DDS_DataWriterQos** (p. 909) values into the provided **DDS_DataWriter↩ Qos** (p. 909) instance.

The retrieved `qos` will match the set of values specified on the last successful call to **DDSDomainParticipant::set↩ _default_datawriter_qos** (p. 1627), or **DDSDomainParticipant::set_default_datawriter_qos_with_profile** (p. 1627), or else, if the call was never made, the default values listed in **DDS_DataWriterQos** (p. 909).

This method may potentially allocate memory depending on the sequences contained in some QoS policies.

MT Safety:

UNSAFE. It is not safe to retrieve the default DataWriterQoS from a DomainPartipant while another thread may be simultaneously calling **DDSDomainParticipant::set_default_datawriter_qos** (p. 1627).

Parameters

qos	<<***inout***>> *(p. 433)* Qos to be filled up.

Exceptions

One	of the **Standard Return Codes** (p. 532)

9.253.2.2 set_default_datawriter_qos()

```
virtual DDS_ReturnCode_t DDSDomainParticipant::set_default_datawriter_qos (
            const DDS_DataWriterQos & qos ) [pure virtual]
```

<<***extension***>> *(p. 432)* Set the default DataWriterQos values for this DomainParticipant.

This set of default values will be inherited for a newly created **DDSPublisher** (p. 1818).

Precondition

The specified QoS policies must be consistent, or else the operation will have no effect and fail with **DDS_RETCODE_INCONSISTENT_POLICY** (p. 533)

MT Safety:

UNSAFE. It is not safe to set the default DataWriter QoS for a DomainParticipant while another thread may be simultaneously calling **DDSDomainParticipant::set_default_datawriter_qos** (p. 1627) or **DDSDomainParticipant::get_default_datawriter_qos** (p. 1626) or calling **DDSDomainParticipant::create_datawriter** (p. 1683) with **DDS_DATAWRITER_QOS_DEFAULT** (p. 114) as the `qos` parameter.

Parameters

qos	<<***in***>> *(p. 433)* Default qos to be set. The special value **DDS_DATAWRITER_QOS_DEFAULT** (p. 114) may be passed as `qos` to indicate that the default QoS should be reset back to the initial values the factory would used if **DDSDomainParticipant::set_default_datawriter_qos** (p. 1627) had never been called.

Exceptions

One	of the **Standard Return Codes** (p. 532), or **DDS_RETCODE_INCONSISTENT_POLICY** (p. 533)

9.253.2.3 set_default_datawriter_qos_with_profile()

```
virtual DDS_ReturnCode_t DDSDomainParticipant::set_default_datawriter_qos_with_profile (
            const char * library_name,
            const char * profile_name ) [pure virtual]
```

<<***extension***>> *(p. 432)* Set the default **DDS_DataWriterQos** (p. 909) values for this domain participant based on the input XML QoS profile.

This set of default values will be inherited for a newly created **DDSPublisher** (p. 1818).

Precondition

The **DDS_DataWriterQos** (p. 909) contained in the specified XML QoS profile must be consistent, or else the operation will have no effect and fail with **DDS_RETCODE_INCONSISTENT_POLICY** (p. 533)

MT Safety:

UNSAFE. It is not safe to set the default DataWriter QoS for a DomainParticipant while another thread may be simultaneously calling **DDSDomainParticipant::set_default_datawriter_qos** (p. 1627) or **DDSDomain↩ Participant::get_default_datawriter_qos** (p. 1626)

Parameters

library_name	<<***in***>> *(p. 433)* Library name containing the XML QoS profile. If library_name is null RTI Connext will use the default library (see **DDSDomainParticipant::set_default_library** (p. 1635)).
profile_name	<<***in***>> *(p. 433)* XML QoS Profile name. If profile_name is null RTI Connext will use the default profile (see **DDSDomainParticipant::set_default_profile** (p. 1636)).

If the input profile cannot be found, the method fails with **DDS_RETCODE_ERROR** (p. 532).

Exceptions

One	of the **Standard Return Codes** (p. 532), or **DDS_RETCODE_INCONSISTENT_POLICY** (p. 533)

9.253.2.4 get_default_datareader_qos()

```
virtual DDS_ReturnCode_t DDSDomainParticipant::get_default_datareader_qos (
            DDS_DataReaderQos & qos ) [pure virtual]
```

<<***extension***>> *(p. 432)* Copy the default **DDS_DataReaderQos** (p. 858) values into the provided **DDS_Data↩ ReaderQos** (p. 858) instance.

The retrieved `qos` will match the set of values specified on the last successful call to **DDSDomainParticipant::set_↩ default_datareader_qos** (p. 1629), or **DDSDomainParticipant::set_default_datareader_qos_with_profile** (p. 1630), or else, if the call was never made, the default values listed in **DDS_DataReaderQos** (p. 858).

This method may potentially allocate memory depending on the sequences contained in some QoS policies.

MT Safety:

UNSAFE. It is not safe to retrieve the default DataReader QoS from a DomainParticipant while another thread may be simultaneously calling **DDSDomainParticipant::set_default_datareader_qos** (p. 1629).

Parameters

qos	<<***inout***>> *(p. 433)* Qos to be filled up.

Exceptions

One	of the **Standard Return Codes** (p. 532)

9.253.2.5 set_default_datareader_qos()

```
virtual DDS_ReturnCode_t DDSDomainParticipant::set_default_datareader_qos (
            const DDS_DataReaderQos & qos ) [pure virtual]
```

<<***extension***>> *(p. 432)* Set the default **DDS_DataReaderQos** (p. 858) values for this domain participant.

This set of default values will be inherited for a newly created **DDSSubscriber** (p. 1860).

Precondition

The specified QoS policies must be consistent, or else the operation will have no effect and fail with **DDS_RETCODE_INCONSISTENT_POLICY** (p. 533)

MT Safety:

UNSAFE. It is not safe to set the default DataReader QoS for a DomainParticipant while another thread may be simultaneously calling **DDSDomainParticipant::set_default_datareader_qos** (p. 1629) or **DDSDomainParticipant::get_default_datareader_qos** (p. 1628).

Parameters

qos	<<***in***>> *(p. 433)* Default qos to be set. The special value **DDS_DATAREADER_QOS_DEFAULT** (p. 136) may be passed as `qos` to indicate that the default QoS should be reset back to the initial values the factory would used if **DDSDomainParticipant::set_default_datareader_qos** (p. 1629) had never been called.

Exceptions

One	of the **Standard Return Codes** (p. 532), or **DDS_RETCODE_INCONSISTENT_POLICY** (p. 533)

9.253.2.6 set_default_datareader_qos_with_profile()

```
virtual DDS_ReturnCode_t DDSDomainParticipant::set_default_datareader_qos_with_profile (
            const char * library_name,
            const char * profile_name ) [pure virtual]
```

<<***extension***>> *(p. 432)* Set the default **DDS_DataReaderQos** (p. 858) values for this DomainParticipant based on the input XML QoS profile.

This set of default values will be inherited for a newly created **DDSSubscriber** (p. 1860).

Precondition

The **DDS_DataReaderQos** (p. 858) contained in the specified XML QoS profile must be consistent, or else the operation will have no effect and fail with **DDS_RETCODE_INCONSISTENT_POLICY** (p. 533)

MT Safety:

UNSAFE. It is not safe to set the default DataReader QoS for a DomainParticipant while another thread may be simultaneously calling **DDSDomainParticipant::set_default_datareader_qos** (p. 1629) or **DDSDomain↩ Participant::get_default_datareader_qos** (p. 1628).

Parameters

library_name	<<***in***>> *(p. 433)* Library name containing the XML QoS profile. If library_name is null RTI Connext will use the default library (see **DDSDomainParticipant::set_default_library** (p. 1635)).
profile_name	<<***in***>> *(p. 433)* XML QoS Profile name. If profile_name is null RTI Connext will use the default profile (see **DDSDomainParticipant::set_default_profile** (p. 1636)).

If the input profile cannot be found, the method fails with **DDS_RETCODE_ERROR** (p. 532).

Exceptions

One	of the **Standard Return Codes** (p. 532), or **DDS_RETCODE_INCONSISTENT_POLICY** (p. 533)

9.253.2.7 get_default_flowcontroller_property()

```
virtual DDS_ReturnCode_t DDSDomainParticipant::get_default_flowcontroller_property (
            DDS_FlowControllerProperty_t & prop ) [pure virtual]
```

<<***extension***>> *(p. 432)* Copies the default **DDS_FlowControllerProperty_t** (p. 1156) values for this domain participant into the given **DDS_FlowControllerProperty_t** (p. 1156) instance.

The retrieved `property` will match the set of values specified on the last successful call to **DDSDomainParticipant::set_default_flowcontroller_property** (p. 1631), or else, if the call was never made, the default values listed in **DDS_FlowControllerProperty_t** (p. 1156).

MT Safety:

UNSAFE. It is not safe to retrieve the default flow controller properties from a DomainParticipant while another thread may be simultaneously calling **DDSDomainParticipant::set_default_flowcontroller_property** (p. 1631)

Parameters

prop	<<***in***>> *(p. 433)* Default property to be retrieved.

Exceptions

One	of the **Standard Return Codes** (p. 532)

See also

DDS_FLOW_CONTROLLER_PROPERTY_DEFAULT (p. 60)

DDSDomainParticipant::create_flowcontroller (p. 1659)

9.253.2.8 set_default_flowcontroller_property()

```
virtual DDS_ReturnCode_t DDSDomainParticipant::set_default_flowcontroller_property (
            const DDS_FlowControllerProperty_t & prop ) [pure virtual]
```

<<***extension***>> *(p. 432)* Set the default **DDS_FlowControllerProperty_t** (p. 1156) values for this domain participant.

This default value will be used for newly created **DDSFlowController** (p. 1736) if **DDS_FLOW_CONTROLLER_PROPERTY_DEFAULT** (p. 60) is specified as the `property` parameter when **DDSDomainParticipant::create_flowcontroller** (p. 1659) is called.

Precondition

The specified property values must be consistent, or else the operation will have no effect and fail with **DDS_RETCODE_INCONSISTENT_POLICY** (p. 533)

MT Safety:

UNSAFE. It is not safe to set the default flow controller properties for a DomainParticipant while another thread may be simultaneously calling **DDSDomainParticipant::set_default_flowcontroller_property** (p. 1631) , **DDSDomainParticipant::get_default_flowcontroller_property** (p. 1630) or calling **DDSDomainParticipant::create_flowcontroller** (p. 1659) with **DDS_FLOW_CONTROLLER_PROPERTY_DEFAULT** (p. 60) as the `qos` parameter.

Parameters

prop	<<***in***>> *(p. 433)* Default property to be set. The special value **DDS_FLOW_CONTROLLER_PROPERTY_DEFAULT** (p. 60) may be passed as `property` to indicate that the default property should be reset to the default values the factory would use if **DDSDomainParticipant::set_default_flowcontroller_property** (p. 1631) had never been called.

Exceptions

One	of the **Standard Return Codes** (p. 532), or **DDS_RETCODE_INCONSISTENT_POLICY** (p. 533)

See also

DDS_FLOW_CONTROLLER_PROPERTY_DEFAULT (p. 60)

DDSDomainParticipant::create_flowcontroller (p. 1659)

9.253.2.9 register_contentfilter()

```
virtual  DDS_ReturnCode_t DDSDomainParticipant::register_contentfilter (
             const char * filter_name,
             const  DDSContentFilter * contentfilter )  [pure virtual]
```

<<***extension***>> *(p. 432)* Register a content filter which can be used to create a **DDSContentFilteredTopic** (p. 1551).

DDS specifies a SQL-like content filter for use by content filtered topics. If this filter does not meet your filtering requirements, you can register a custom filter.

To use a custom filter, it must be registered in the following places:

- In any application that uses the custom filter to create a **DDSContentFilteredTopic** (p. 1551) and the corresponding **DDSDataReader** (p. 1556).
- In each application that writes the data to the applications mentioned above.

For example, suppose Application A on the subscription side creates a Topic named X and a ContentFilteredTopic named filteredX (and a corresponding DataReader), using a previously registered content filter, myFilter. With only that, you will have filtering at the subscription side. If you also want to perform filtering in any application that publishes Topic X, then you also need to register the same definition of the ContentFilter myFilter in that application.

Each `filter_name` can only be used to registered a content filter once with a **DDSDomainParticipant** (p. 1619).

Parameters

filter_name	<<***in***>> *(p. 433)* Name of the filter. The name must be unique within the **DDSDomainParticipant** (p. 1619) and must not exceed 255 characters. Cannot be NULL.
contentfilter	<<***in***>> *(p. 433)* Content filter to be registered. Cannot be NULL.

Exceptions

One	of the **Standard Return Codes** (p. 532)

See also

DDSDomainParticipant::unregister_contentfilter (p. 1633)

9.253.2.10 lookup_contentfilter()

```
virtual DDSContentFilter * DDSDomainParticipant::lookup_contentfilter (
            const char * filter_name ) [pure virtual]
```

<<***extension***>> *(p. 432)* Lookup a content filter previously registered with **DDSDomainParticipant::register_↩ contentfilter** (p. 1632).

Parameters

filter_name	<<***in***>> *(p. 433)* Name of the filter. Cannot be NULL.

Returns

NULL if the given filter_name has not been previously registered to the **DDSDomainParticipant** (p. 1619) with **DDSDomainParticipant::register_contentfilter** (p. 1632). Otherwise, return the **DDSContentFilter** (p. 1548) that has been previosuly registered with the given filter_name.

See also

DDSDomainParticipant::register_contentfilter (p. 1632)

9.253.2.11 unregister_contentfilter()

```
virtual DDS_ReturnCode_t DDSDomainParticipant::unregister_contentfilter (
            const char * filter_name ) [pure virtual]
```

<<***extension***>> *(p. 432)* Unregister a content filter previously registered with **DDSDomainParticipant::register_↩ contentfilter** (p. 1632).

A `filter_name` can be unregistered only if it has been previously registered to the **DDSDomainParticipant** (p. 1619) with **DDSDomainParticipant::register_contentfilter** (p. 1632).

The unregistration of filter is not allowed if there are any existing **DDSContentFilteredTopic** (p. 1551) objects that are using the filter. If the operation is called on a filter with existing **DDSContentFilteredTopic** (p. 1551) objects attached to it, this operation will fail with **DDS_RETCODE_PRECONDITION_NOT_MET** (p. 532).

If there are still existing discovered **DDSDataReader** (p. 1556) s with the same `filter_name` and the filter's compile method of the filter have previously been called on the discovered **DDSDataReader** (p. 1556) s, finalize method of the filter will be called on those discovered **DDSDataReader** (p. 1556) s before the content filter is unregistered. This means filtering will now be performed on the application that is creating the **DDSDataReader** (p. 1556).

Parameters

filter_name	<<***in***>> *(p. 433)* Name of the filter. Cannot be NULL.

Exceptions

One	of the **Standard Return Codes** (p. 532) or **DDS_RETCODE_PRECONDITION_NOT_MET** (p. 532)

See also

DDSDomainParticipant::register_contentfilter (p. 1632)

9.253.2.12 get_default_library()

```
virtual const char * DDSDomainParticipant::get_default_library ( )  [pure virtual]
```

<<***extension***>> *(p. 432)* Gets the default XML library associated with a **DDSDomainParticipant** (p. 1619).

Returns

The default library or null if the default library was not set.

See also

DDSDomainParticipant::set_default_library (p. 1635)

9.253.2.13 get_default_profile()

```
virtual const char * DDSDomainParticipant::get_default_profile ( )  [pure virtual]
```

<<***extension***>> *(p. 432)* Gets the default XML profile associated with a **DDSDomainParticipant** (p. 1619).

Returns

The default profile or null if the default profile was not set.

See also

DDSDomainParticipant::set_default_profile (p. 1636)

9.253.2.14 get_default_profile_library()

```
virtual const char * DDSDomainParticipant::get_default_profile_library ( )  [pure virtual]
```

<<***extension***>> *(p. 432)* Gets the library where the default XML QoS profile is contained for a **DDSDomainParticipant** (p. 1619).

The default profile library is automatically set when **DDSDomainParticipant::set_default_profile** (p. 1636) is called.

This library can be different than the **DDSDomainParticipant** (p. 1619) default library (see **DDSDomainParticipant::get_default_library** (p. 1634)).

Returns

The default profile library or null if the default profile was not set.

See also

DDSDomainParticipant::set_default_profile (p. 1636)

9.253.2.15 set_default_library()

```
virtual  DDS_ReturnCode_t DDSDomainParticipant::set_default_library (
            const char * library_name )  [pure virtual]
```

<<***extension***>> *(p. 432)* Sets the default XML library for a **DDSDomainParticipant** (p. 1619).

This method specifies the library that will be used as the default the next time a default library is needed during a call to one of this DomainParticipant's operations.

Any API requiring a library_name as a parameter can use null to refer to the default library.

If the default library is not set, the **DDSDomainParticipant** (p. 1619) inherits the default from the **DDSDomainParticipantFactory** (p. 1694) (see **DDSDomainParticipantFactory::set_default_library** (p. 1701)).

Parameters

library_name	<<***in***>> *(p. 433)* Library name. If library_name is null any previous default is unset.

Exceptions

One	of the **Standard Return Codes** (p. 532)

See also

DDSDomainParticipant::get_default_library (p. 1634)

9.253.2.16 set_default_profile()

```
virtual DDS_ReturnCode_t DDSDomainParticipant::set_default_profile (
            const char * library_name,
            const char * profile_name ) [pure virtual]
```

<<***extension***>> *(p. 432)* Sets the default XML profile for a **DDSDomainParticipant** (p. 1619).

This method specifies the profile that will be used as the default the next time a default DomainParticipant profile is needed during a call to one of this DomainParticipant's operations. When calling a **DDSDomainParticipant** (p. 1619) method that requires a `profile_name` parameter, you can use NULL to refer to the default profile. (This same information applies to setting a default library.)

If the default profile is not set, the **DDSDomainParticipant** (p. 1619) inherits the default from the **DDSDomain↩ ParticipantFactory** (p. 1694) (see **DDSDomainParticipantFactory::set_default_profile** (p. 1702)).

This method does not set the default QoS for entities created by the **DDSDomainParticipant** (p. 1619); for this functionality, use the methods set_default_<entity>_qos_with_profile (you may pass in NULL after having called **set_default↩ _profile()** (p. 1636)).

This method does not set the default QoS for newly created DomainParticipants; for this functionality, use **DDSDomain↩ ParticipantFactory::set_default_participant_qos_with_profile** (p. 1699).

Parameters

library_name	<<***in***>> *(p. 433)* The library name containing the profile.
profile_name	<<***in***>> *(p. 433)* The profile name. If profile_name is null any previous default is unset.

Exceptions

One	of the **Standard Return Codes** (p. 532)

See also

DDSDomainParticipant::get_default_profile (p. 1634)

DDSDomainParticipant::get_default_profile_library (p. 1634)

9.253.2.17 get_default_topic_qos()

```
virtual DDS_ReturnCode_t DDSDomainParticipant::get_default_topic_qos (
            DDS_TopicQos & qos ) [pure virtual]
```

Copies the default **DDS_TopicQos** (p. 1397) values for this domain participant into the given **DDS_TopicQos** (p. 1397) instance.

The retrieved `qos` will match the set of values specified on the last successful call to **DDSDomainParticipant::set_default_topic_qos** (p. 1637), or **DDSDomainParticipant::set_default_topic_qos_with_profile** (p. 1638) or else, if the call was never made, the default values listed in **DDS_TopicQos** (p. 1397).

This method may potentially allocate memory depending on the sequences contained in some QoS policies.

MT Safety:

UNSAFE. It is not safe to retrieve the default Topic QoS from a DomainParticipant while another thread may be simultaneously calling **DDSDomainParticipant::set_default_topic_qos** (p. 1637)

Parameters

qos	<<***in***>> *(p. 433)* Default qos to be retrieved.

Exceptions

One	of the **Standard Return Codes** (p. 532)

See also

DDS_TOPIC_QOS_DEFAULT (p. 58)

DDSDomainParticipant::create_topic (p. 1650)

9.253.2.18 set_default_topic_qos()

```
virtual DDS_ReturnCode_t DDSDomainParticipant::set_default_topic_qos (
            const DDS_TopicQos & qos ) [pure virtual]
```

Set the default **DDS_TopicQos** (p. 1397) values for this domain participant.

This default value will be used for newly created **DDSTopic** (p. 1885) if **DDS_TOPIC_QOS_DEFAULT** (p. 58) is specified as the `qos` parameter when **DDSDomainParticipant::create_topic** (p. 1650) is called.

Precondition

The specified QoS policies must be consistent, or else the operation will have no effect and fail with **DDS_RETCODE_INCONSISTENT_POLICY** (p. 533)

MT Safety:

UNSAFE. It is not safe to set the default topic QoS for a DomainParticipant while another thread may be simultaneously calling **DDSDomainParticipant::set_default_topic_qos** (p. 1637), **DDSDomainParticipant::get_default_topic_qos** (p. 1636) or calling **DDSDomainParticipant::create_topic** (p. 1650) with **DDS_TOPIC_QOS_DEFAULT** (p. 58) as the `qos` parameter.

Parameters

qos	<<***in***>> *(p. 433)* Default qos to be set. The special value **DDS_TOPIC_QOS_DEFAULT** (p. 58) may be passed as `qos` to indicate that the default QoS should be reset back to the initial values the factory would used if **DDSDomainParticipant::set_default_topic_qos** (p. 1637) had never been called.

Exceptions

One	of the **Standard Return Codes** (p. 532), or **DDS_RETCODE_INCONSISTENT_POLICY** (p. 533)

See also

DDS_TOPIC_QOS_DEFAULT (p. 58)

DDSDomainParticipant::create_topic (p. 1650)

9.253.2.19 set_default_topic_qos_with_profile()

```
virtual DDS_ReturnCode_t DDSDomainParticipant::set_default_topic_qos_with_profile (
            const char * library_name,
            const char * profile_name ) [pure virtual]
```

<<***extension***>> *(p. 432)* Set the default **DDS_TopicQos** (p. 1397) values for this domain participant based on the input XML QoS profile.

This default value will be used for newly created **DDSTopic** (p. 1885) if **DDS_TOPIC_QOS_DEFAULT** (p. 58) is specified as the `qos` parameter when **DDSDomainParticipant::create_topic** (p. 1650) is called.

Precondition

The **DDS_TopicQos** (p. 1397) contained in the specified XML QoS profile must be consistent, or else the operation will have no effect and fail with **DDS_RETCODE_INCONSISTENT_POLICY** (p. 533)

MT Safety:

UNSAFE. It is not safe to set the default topic QoS for a DomainParticipant while another thread may be simultaneously calling **DDSDomainParticipant::set_default_topic_qos** (p. 1637), **DDSDomainParticipant::get_default_topic_qos** (p. 1636) or calling **DDSDomainParticipant::create_topic** (p. 1650) with **DDS_TOPIC_QOS_DEFAULT** (p. 58) as the `qos` parameter.

Parameters

library_name	<<***in***>> *(p. 433)* Library name containing the XML QoS profile. If library_name is null RTI Connext will use the default library (see **DDSDomainParticipant::set_default_library** (p. 1635)).
profile_name	<<***in***>> *(p. 433)* XML QoS Profile name. If profile_name is null RTI Connext will use the default profile (see **DDSDomainParticipant::set_default_profile** (p. 1636)).

If the input profile cannot be found the method fails with **DDS_RETCODE_ERROR** (p. 532).

Exceptions

One	of the **Standard Return Codes** (p. 532), or **DDS_RETCODE_INCONSISTENT_POLICY** (p. 533)

See also

DDS_TOPIC_QOS_DEFAULT (p. 58)

DDSDomainParticipant::create_topic_with_profile (p. 1651)

9.253.2.20 get_default_publisher_qos()

```
virtual DDS_ReturnCode_t DDSDomainParticipant::get_default_publisher_qos (
            DDS_PublisherQos & qos ) [pure virtual]
```

Copy the default **DDS_PublisherQos** (p. 1276) values into the provided **DDS_PublisherQos** (p. 1276) instance.

The retrieved `qos` will match the set of values specified on the last successful call to **DDSDomainParticipant::set_default_publisher_qos** (p. 1640), or **DDSDomainParticipant::set_default_publisher_qos_with_profile** (p. 1640), or else, if the call was never made, the default values listed in **DDS_PublisherQos** (p. 1276).

This method may potentially allocate memory depending on the sequences contained in some QoS policies.

If **DDS_PUBLISHER_QOS_DEFAULT** (p. 59) is specified as the `qos` parameter when **DDSDomainParticipant::create_topic** (p. 1650) is called, the default value of the QoS set in the factory, equivalent to the value obtained by calling **DDSDomainParticipant::get_default_publisher_qos** (p. 1639), will be used to create the **DDSPublisher** (p. 1818).

MT Safety:

UNSAFE. It is not safe to retrieve the default publisher QoS from a DomainParticipant while another thread may be simultaneously calling **DDSDomainParticipant::set_default_publisher_qos** (p. 1640)

Parameters

qos	<<***inout***>> *(p. 433)* Qos to be filled up.

Exceptions

One	of the **Standard Return Codes** (p. 532)

See also

DDS_PUBLISHER_QOS_DEFAULT (p. 59)

DDSDomainParticipant::create_publisher (p. 1644)

9.253.2.21 set_default_publisher_qos()

```
virtual DDS_ReturnCode_t DDSDomainParticipant::set_default_publisher_qos (
            const DDS_PublisherQos & qos ) [pure virtual]
```

Set the default **DDS_PublisherQos** (p. 1276) values for this DomainParticipant.

This set of default values will be used for a newly created **DDSPublisher** (p. 1818) if **DDS_PUBLISHER_QOS_↩ DEFAULT** (p. 59) is specified as the qos parameter when **DDSDomainParticipant::create_publisher** (p. 1644) is called.

Precondition

The specified QoS policies must be consistent, or else the operation will have no effect and fail with **DDS_↩ RETCODE_INCONSISTENT_POLICY** (p. 533)

MT Safety:

UNSAFE. It is not safe to set the default publisher QoS for a DomainParticipant while another thread may be simultaneously calling **DDSDomainParticipant::set_default_publisher_qos** (p. 1640), **DDSDomainParticipant↩ ::get_default_publisher_qos** (p. 1639) or calling **DDSDomainParticipant::create_publisher** (p. 1644) with **DDS_PUBLISHER_QOS_DEFAULT** (p. 59) as the qos parameter.

Parameters

qos	<<***in***>> *(p. 433)* Default qos to be set. The special value **DDS_PUBLISHER_QOS_DEFAULT** (p. 59) may be passed as qos to indicate that the default QoS should be reset back to the initial values the factory would used if **DDSDomainParticipant::set_default_publisher_qos** (p. 1640) had never been called.

Exceptions

One	of the **Standard Return Codes** (p. 532), or **DDS_RETCODE_INCONSISTENT_POLICY** (p. 533)

See also

DDS_PUBLISHER_QOS_DEFAULT (p. 59)

DDSDomainParticipant::create_publisher (p. 1644)

9.253.2.22 set_default_publisher_qos_with_profile()

```
virtual  DDS_ReturnCode_t DDSDomainParticipant::set_default_publisher_qos_with_profile (
            const char * library_name,
            const char * profile_name )  [pure virtual]
```

<<***extension***>> *(p. 432)* Set the default **DDS_PublisherQos** (p. 1276) values for this DomainParticipant based on the input XML QoS profile.

This set of default values will be used for a newly created **DDSPublisher** (p. 1818) if **DDS_PUBLISHER_QOS_↩ DEFAULT** (p. 59) is specified as the `qos` parameter when **DDSDomainParticipant::create_publisher** (p. 1644) is called.

Precondition

The **DDS_PublisherQos** (p. 1276) contained in the specified XML QoS profile must be consistent, or else the operation will have no effect and fail with **DDS_RETCODE_INCONSISTENT_POLICY** (p. 533)

MT Safety:

UNSAFE. It is not safe to set the default publisher QoS for a DomainParticipant while another thread may be simultaneously calling **DDSDomainParticipant::set_default_publisher_qos** (p. 1640), **DDSDomainParticipant↩ ::get_default_publisher_qos** (p. 1639) or calling **DDSDomainParticipant::create_publisher** (p. 1644) with **DDS_PUBLISHER_QOS_DEFAULT** (p. 59) as the `qos` parameter.

Parameters

library_name	<<***in***>> *(p. 433)* Library name containing the XML QoS profile. If library_name is null RTI Connext will use the default library (see **DDSDomainParticipant::set_default_library** (p. 1635)).
profile_name	<<***in***>> *(p. 433)* XML QoS Profile name. If profile_name is null RTI Connext will use the default profile (see **DDSDomainParticipant::set_default_profile** (p. 1636)).

If the input profile cannot be found, the method fails with **DDS_RETCODE_ERROR** (p. 532).

Exceptions

One	of the **Standard Return Codes** (p. 532), or **DDS_RETCODE_INCONSISTENT_POLICY** (p. 533)

See also

DDS_PUBLISHER_QOS_DEFAULT (p. 59)

DDSDomainParticipant::create_publisher_with_profile (p. 1645)

9.253.2.23 get_default_subscriber_qos()

```
virtual DDS_ReturnCode_t DDSDomainParticipant::get_default_subscriber_qos (
            DDS_SubscriberQos & qos ) [pure virtual]
```

Copy the default **DDS_SubscriberQos** (p. 1367) values into the provided **DDS_SubscriberQos** (p. 1367) instance.

The retrieved `qos` will match the set of values specified on the last successful call to **DDSDomainParticipant::set_default_subscriber_qos** (p. 1642), or **DDSDomainParticipant::set_default_subscriber_qos_with_profile** (p. 1643), or else, if the call was never made, the default values listed in **DDS_SubscriberQos** (p. 1367).

This method may potentially allocate memory depending on the sequences contained in some QoS policies.

If **DDS_SUBSCRIBER_QOS_DEFAULT** (p. 59) is specified as the `qos` parameter when **DDSDomainParticipant::create_subscriber** (p. 1647) is called, the default value of the QoS set in the factory, equivalent to the value obtained by calling **DDSDomainParticipant::get_default_subscriber_qos** (p. 1641), will be used to create the **DDSSubscriber** (p. 1860).

MT Safety:

UNSAFE. It is not safe to retrieve the default Subscriber QoS from a DomainParticipant while another thread may be simultaneously calling **DDSDomainParticipant::set_default_subscriber_qos** (p. 1642).

Parameters

qos	<<***inout***>> *(p. 433)* Qos to be filled up.

Exceptions

One	of the **Standard Return Codes** (p. 532)

See also

DDS_SUBSCRIBER_QOS_DEFAULT (p. 59)

DDSDomainParticipant::create_subscriber (p. 1647)

9.253.2.24 set_default_subscriber_qos()

```
virtual DDS_ReturnCode_t DDSDomainParticipant::set_default_subscriber_qos (
            const DDS_SubscriberQos & qos ) [pure virtual]
```

Set the default **DDS_SubscriberQos** (p. 1367) values for this DomainParticipant.

This set of default values will be used for a newly created **DDSSubscriber** (p. 1860) if **DDS_SUBSCRIBER_QOS_DEFAULT** (p. 59) is specified as the `qos` parameter when **DDSDomainParticipant::create_subscriber** (p. 1647) is called.

Precondition

The specified QoS policies must be consistent, or else the operation will have no effect and fail with **DDS_RETCODE_INCONSISTENT_POLICY** (p. 533)

MT Safety:

UNSAFE. It is not safe to set the default Subscriber QoS for a DomainParticipant while another thread may be simultaneously calling **DDSDomainParticipant::set_default_subscriber_qos** (p. 1642), **DDSDomainParticipant::get_default_subscriber_qos** (p. 1641) or calling **DDSDomainParticipant::create_subscriber** (p. 1647) with **DDS_SUBSCRIBER_QOS_DEFAULT** (p. 59) as the `qos` parameter.

Parameters

qos	<<***in***>> *(p. 433)* Default qos to be set. The special value **DDS_SUBSCRIBER_QOS_DEFAULT** (p. 59) may be passed as `qos` to indicate that the default QoS should be reset back to the initial values the factory would used if **DDSDomainParticipant::set_default_subscriber_qos** (p. 1642) had never been called.

Exceptions

One	of the **Standard Return Codes** (p. 532), or **DDS_RETCODE_INCONSISTENT_POLICY** (p. 533)

9.253.2.25 set_default_subscriber_qos_with_profile()

```
virtual DDS_ReturnCode_t DDSDomainParticipant::set_default_subscriber_qos_with_profile (
            const char * library_name,
            const char * profile_name ) [pure virtual]
```

<<***extension***>> *(p. 432)* Set the default **DDS_SubscriberQos** (p. 1367) values for this DomainParticipant based on the input XML QoS profile.

This set of default values will be used for a newly created **DDSSubscriber** (p. 1860) if **DDS_SUBSCRIBER_QOS_DEFAULT** (p. 59) is specified as the `qos` parameter when **DDSDomainParticipant::create_subscriber** (p. 1647) is called.

Precondition

The **DDS_SubscriberQos** (p. 1367) contained in the specified XML QoS profile must be consistent, or else the operation will have no effect and fail with **DDS_RETCODE_INCONSISTENT_POLICY** (p. 533)

MT Safety:

UNSAFE. It is not safe to set the default Subscriber QoS for a DomainParticipant while another thread may be simultaneously calling **DDSDomainParticipant::set_default_subscriber_qos** (p. 1642), **DDSDomainParticipant::get_default_subscriber_qos** (p. 1641) or calling **DDSDomainParticipant::create_subscriber** (p. 1647) with **DDS_SUBSCRIBER_QOS_DEFAULT** (p. 59) as the `qos` parameter.

Parameters

library_name	<<***in***>> *(*p. *433)* Library name containing the XML QoS profile. If library_name is null RTI Connext will use the default library (see **DDSDomainParticipant::set_default_library** (p. 1635)).
profile_name	<<***in***>> *(*p. *433)* XML QoS Profile name. If profile_name is null RTI Connext will use the default profile (see **DDSDomainParticipant::set_default_profile** (p. 1636)).

If the input profile cannot be found, the method fails with **DDS_RETCODE_ERROR** (p. 532).

Exceptions

One	of the **Standard Return Codes** (p. 532), or **DDS_RETCODE_INCONSISTENT_POLICY** (p. 533)

See also

DDS_SUBSCRIBER_QOS_DEFAULT (p. 59)

DDSDomainParticipant::create_subscriber_with_profile (p. 1648)

9.253.2.26 create_publisher()

```
virtual DDSPublisher * DDSDomainParticipant::create_publisher (
            const DDS_PublisherQos & qos,
             DDSPublisherListener * listener,
             DDS_StatusMask mask ) [pure virtual]
```

Creates a **DDSPublisher** (p. 1818) with the desired QoS policies and attaches to it the specified **DDSPublisherListener** (p. 1839).

Precondition

The specified QoS policies must be consistent, or the operation will fail and no **DDSPublisher** (p. 1818) will be created.

MT Safety:

UNSAFE. If **DDS_PUBLISHER_QOS_DEFAULT** (p. 59) is used for `qos`, it is not safe to create the publisher while another thread may be simultaneously calling **DDSDomainParticipant::set_default_publisher_qos** (p. 1640).

Parameters

qos	<<***in***>> *(*p. *433)* QoS to be used for creating the new **DDSPublisher** (p. 1818). The special value **DDS_PUBLISHER_QOS_DEFAULT** (p. 59) can be used to indicate that the **DDSPublisher** (p. 1818) should be created with the default **DDS_PublisherQos** (p. 1276) set in the **DDSDomainParticipant** (p. 1619).
listener	<<***in***>> *(*p. *433)*. Listener to be attached to the newly created **DDSPublisher** (p. 1818).
mask	<<***in***>> *(*p. *433)*. Changes of communication status to be invoked on the listener. See **DDS_StatusMask** (p. 537).

Returns

newly created publisher object or NULL on failure.

See also

Specifying QoS on entities (p. 555) for information on setting QoS before entity creation

DDS_PublisherQos (p. 1276) for rules on consistency among QoS

DDS_PUBLISHER_QOS_DEFAULT (p. 59)

DDSDomainParticipant::create_publisher_with_profile (p. 1645)

DDSDomainParticipant::get_default_publisher_qos (p. 1639)

DDSPublisher::set_listener (p. 1838)

Examples

HelloWorld_publisher.cxx.

9.253.2.27 create_publisher_with_profile()

```
virtual  DDSPublisher * DDSDomainParticipant::create_publisher_with_profile (
             const char * library_name,
             const char * profile_name,
              DDSPublisherListener * listener,
              DDS_StatusMask mask )  [pure virtual]
```

<<***extension***>> *(p. 432)* Creates a new **DDSPublisher** (p. 1818) object using the **DDS_PublisherQos** (p. 1276) associated with the input XML QoS profile.

Precondition

The **DDS_PublisherQos** (p. 1276) in the input profile must be consistent, or the operation will fail and no **DDSPublisher** (p. 1818) will be created.

Parameters

library_name	<<***in***>> *(p. 433)* Library name containing the XML QoS profile. If library_name is null RTI Connext will use the default library (see **DDSDomainParticipant::set_default_library** (p. 1635)).
profile_name	<<***in***>> *(p. 433)* XML QoS Profile name. If profile_name is null RTI Connext will use the default profile (see **DDSDomainParticipant::set_default_profile** (p. 1636)).
listener	<<***in***>> *(p. 433).* Listener to be attached to the newly created **DDSPublisher** (p. 1818).
mask	<<***in***>> *(p. 433).* Changes of communication status to be invoked on the listener. See **DDS_StatusMask** (p. 537).

Returns

newly created publisher object or NULL on failure.

See also

Specifying QoS on entities (p. 555) for information on setting QoS before entity creation

DDS_PublisherQos (p. 1276) for rules on consistency among QoS

DDSDomainParticipant::create_publisher (p. 1644)

DDSDomainParticipant::get_default_publisher_qos (p. 1639)

DDSPublisher::set_listener (p. 1838)

9.253.2.28 delete_publisher()

```
virtual DDS_ReturnCode_t DDSDomainParticipant::delete_publisher (
            DDSPublisher * p ) [pure virtual]
```

Deletes an existing **DDSPublisher** (p. 1818).

Precondition

The **DDSPublisher** (p. 1818) must not have any attached **DDSDataWriter** (p. 1589) objects. If there are existing **DDSDataWriter** (p. 1589) objects, it will fail with **DDS_RETCODE_PRECONDITION_NOT_MET** (p. 532).

DDSPublisher (p. 1818) must have been created by this **DDSDomainParticipant** (p. 1619), or else it will fail with **DDS_RETCODE_PRECONDITION_NOT_MET** (p. 532).

Postcondition

Listener installed on the **DDSPublisher** (p. 1818) will not be called after this method completes successfully.

MT Safety:

UNSAFE. It is not safe to delete an entity while another thread may be simultaneously calling an API that uses the entity.

Parameters

p	<<***in***>> *(p. 433)* **DDSPublisher** (p. 1818) to be deleted.

Exceptions

One	of the **Standard Return Codes** (p. 532), or **DDS_RETCODE_PRECONDITION_NOT_MET** (p. 532).

9.253.2.29 create_subscriber()

```
virtual DDSSubscriber * DDSDomainParticipant::create_subscriber (
            const DDS_SubscriberQos & qos,
             DDSSubscriberListener * listener,
             DDS_StatusMask mask ) [pure virtual]
```

Creates a **DDSSubscriber** (p. 1860) with the desired QoS policies and attaches to it the specified **DDSSubscriberListener** (p. 1881).

Precondition

The specified QoS policies must be consistent, or the operation will fail and no **DDSSubscriber** (p. 1860) will be created.

MT Safety:

UNSAFE. If **DDS_SUBSCRIBER_QOS_DEFAULT** (p. 59) is used for `qos`, it is not safe to create the subscriber while another thread may be simultaneously calling **DDSDomainParticipant::set_default_subscriber_qos** (p. 1642).

Parameters

qos	<<***in***>> *(p. 433)* QoS to be used for creating the new **DDSSubscriber** (p. 1860). The special value **DDS_SUBSCRIBER_QOS_DEFAULT** (p. 59) can be used to indicate that the **DDSSubscriber** (p. 1860) should be created with the default **DDS_SubscriberQos** (p. 1367) set in the **DDSDomainParticipant** (p. 1619).
listener	<<***in***>> *(p. 433)*. Listener to be attached to the newly created **DDSSubscriber** (p. 1860).
mask	<<***in***>> *(p. 433)*. Changes of communication status to be invoked on the listener. See **DDS_StatusMask** (p. 537).

Returns

newly created subscriber object or NULL on failure.

See also

Specifying QoS on entities (p. 555) for information on setting QoS before entity creation

DDS_SubscriberQos (p. 1367) for rules on consistency among QoS

DDS_SUBSCRIBER_QOS_DEFAULT (p. 59)

DDSDomainParticipant::create_subscriber_with_profile (p. 1648)

DDSDomainParticipant::get_default_subscriber_qos (p. 1641)

DDSSubscriber::set_listener (p. 1879)

Examples

HelloWorld_subscriber.cxx.

9.253.2.30 create_subscriber_with_profile()

```
virtual  DDSSubscriber * DDSDomainParticipant::create_subscriber_with_profile (
             const char * library_name,
             const char * profile_name,
              DDSSubscriberListener * listener,
              DDS_StatusMask mask )  [pure virtual]
```

<<***extension***>> *(*p. *432)* Creates a new **DDSSubscriber** (p. 1860) object using the **DDS_PublisherQos** (p. 1276) associated with the input XML QoS profile.

Precondition

The **DDS_SubscriberQos** (p. 1367) in the input profile must be consistent, or the operation will fail and no **DDSSubscriber** (p. 1860) will be created.

Parameters

library_name	<<***in***>> *(*p. *433)* Library name containing the XML QoS profile. If library_name is null RTI Connext will use the default library (see **DDSDomainParticipant::set_default_library** (p. 1635)).
profile_name	<<***in***>> *(*p. *433)* XML QoS Profile name. If profile_name is null RTI Connext will use the default profile (see **DDSDomainParticipant::set_default_profile** (p. 1636)).
listener	<<***in***>> *(*p. *433)*. Listener to be attached to the newly created **DDSSubscriber** (p. 1860).
mask	<<***in***>> *(*p. *433)*. Changes of communication status to be invoked on the listener. See **DDS_StatusMask** (p. 537).

Returns

newly created subscriber object or NULL on failure.

See also

Specifying QoS on entities (p. 555) for information on setting QoS before entity creation

DDS_SubscriberQos (p. 1367) for rules on consistency among QoS

DDSDomainParticipant::create_subscriber (p. 1647)

DDSDomainParticipant::get_default_subscriber_qos (p. 1641)

DDSSubscriber::set_listener (p. 1879)

9.253.2.31 delete_subscriber()

```
virtual DDS_ReturnCode_t DDSDomainParticipant::delete_subscriber (
            DDSSubscriber * s ) [pure virtual]
```

Deletes an existing **DDSSubscriber** (p. 1860).

Precondition

The **DDSSubscriber** (p. 1860) must not have any attached **DDSDataReader** (p. 1556) objects. If there are existing **DDSDataReader** (p. 1556) objects, it will fail with **DDS_RETCODE_PRECONDITION_NOT_MET** (p. 532)

The **DDSSubscriber** (p. 1860) must have been created by this **DDSDomainParticipant** (p. 1619), or else it will fail with **DDS_RETCODE_PRECONDITION_NOT_MET** (p. 532).

Postcondition

A Listener installed on the **DDSSubscriber** (p. 1860) will not be called after this method completes successfully.

MT Safety:

UNSAFE. It is not safe to delete an entity while another thread may be simultaneously calling an API that uses the entity.

Parameters

s	<<***in***>> *(p. 433)* **DDSSubscriber** (p. 1860) to be deleted.

Exceptions

One	of the **Standard Return Codes** (p. 532), or **DDS_RETCODE_PRECONDITION_NOT_MET** (p. 532).

9.253.2.32 get_publishers()

```
virtual DDS_ReturnCode_t DDSDomainParticipant::get_publishers (
            DDSPublisherSeq & publishers ) [pure virtual]
```

<<***extension***>> *(p. 432)* Allows the application to access all the publishers the participant has.

If the sequence doesn't own its buffer, and its maximum is less than the total number of publishers, it will be filled up to its maximum, and fail with **DDS_RETCODE_OUT_OF_RESOURCES** (p. 533).

MT Safety:

Safe.

Parameters

publishers	<<***inout***>> *(p. 433)* a PublisherSeq object where the set or list of publishers will be returned

Exceptions

One	of the **Standard Return Codes** (p. 532), **DDS_RETCODE_OUT_OF_RESOURCES** (p. 533)

9.253.2.33 get_subscribers()

```
virtual  DDS_ReturnCode_t DDSDomainParticipant::get_subscribers (
             DDSSubscriberSeq & subscribers )  [pure virtual]
```

<<***extension***>> *(p. 432)* Allows the application to access all the subscribers the participant has.

If the sequence doesn't own its buffer, and its maximum is less than the total number of subscribers, it will be filled up to its maximum, and fail with **DDS_RETCODE_OUT_OF_RESOURCES** (p. 533).

MT Safety:

Safe.

Parameters

subscribers	<<***inout***>> *(p. 433)* a SubscriberSeq object where the set or list of subscribers will be returned

Exceptions

One	of the **Standard Return Codes** (p. 532), **DDS_RETCODE_OUT_OF_RESOURCES** (p. 533)

9.253.2.34 create_topic()

```
virtual  DDSTopic * DDSDomainParticipant::create_topic (
             const char * topic_name,
             const char * type_name,
             const  DDS_TopicQos & qos,
              DDSTopicListener * listener,
              DDS_StatusMask mask )  [pure virtual]
```

Creates a **DDSTopic** (p. 1885) with the desired QoS policies and attaches to it the specified **DDSTopicListener** (p. 1894).

Precondition

The application is not allowed to create two **DDSTopic** (p. 1885) objects with the same `topic_name` attached to the same **DDSDomainParticipant** (p. 1619). If the application attempts this, this method will fail and return a NULL topic.

The specified QoS policies must be consistent, or the operation will fail and no **DDSTopic** (p. 1885) will be created.

Prior to creating a **DDSTopic** (p. 1885), the type must have been registered with RTI Connext. This is done using the **FooTypeSupport::register_type** (p. 1981) operation on a derived class of the **DDSTypeSupport** (p. 1897) interface.

MT Safety:

UNSAFE. It is not safe to create a topic while another thread is trying to lookup that topic description with **DDSDomainParticipant::lookup_topicdescription** (p. 1658).

MT Safety:

UNSAFE. If **DDS_TOPIC_QOS_DEFAULT** (p. 58) is used for `qos`, it is not safe to create the topic while another thread may be simultaneously calling **DDSDomainParticipant::set_default_topic_qos** (p. 1637).

Parameters

topic_name	<<***in***>> *(p. 433)* Name for the new topic, must not exceed 255 characters. Cannot be NULL.
type_name	<<***in***>> *(p. 433)* The type to which the new **DDSTopic** (p. 1885) will be bound. Cannot be NULL.
qos	<<***in***>> *(p. 433)* QoS to be used for creating the new **DDSTopic** (p. 1885). The special value **DDS_TOPIC_QOS_DEFAULT** (p. 58) can be used to indicate that the **DDSTopic** (p. 1885) should be created with the default **DDS_TopicQos** (p. 1397) set in the **DDSDomainParticipant** (p. 1619).
listener	<<***in***>> *(p. 433)*. Listener to be attached to the newly created **DDSTopic** (p. 1885).
mask	<<***in***>> *(p. 433)*. Changes of communication status to be invoked on the listener. See **DDS_StatusMask** (p. 537).

Returns

newly created topic, or NULL on failure

See also

Specifying QoS on entities (p. 555) for information on setting QoS before entity creation

DDS_TopicQos (p. 1397) for rules on consistency among QoS

DDS_TOPIC_QOS_DEFAULT (p. 58)

DDSDomainParticipant::create_topic_with_profile (p. 1651)

DDSDomainParticipant::get_default_topic_qos (p. 1636)

DDSTopic::set_listener (p. 1889)

Examples

HelloWorld_publisher.cxx, and **HelloWorld_subscriber.cxx**.

9.253.2.35 create_topic_with_profile()

```
virtual  DDSTopic * DDSDomainParticipant::create_topic_with_profile (
            const char * topic_name,
            const char * type_name,
            const char * library_name,
            const char * profile_name,
             DDSTopicListener * listener,
             DDS_StatusMask mask )  [pure virtual]
```

<<***extension***>> *(p. 432)* Creates a new **DDSTopic** (p. 1885) object using the **DDS_PublisherQos** (p. 1276) associated with the input XML QoS profile.

Precondition

The application is not allowed to create two **DDSTopicDescription** (p. 1892) objects with the same `topic_name` attached to the same **DDSDomainParticipant** (p. 1619). If the application attempts this, this method will fail and return a NULL topic.

The **DDS_TopicQos** (p. 1397) in the input profile must be consistent, or the operation will fail and no **DDSTopic** (p. 1885) will be created.

Prior to creating a **DDSTopic** (p. 1885), the type must have been registered with RTI Connext. This is done using the **FooTypeSupport::register_type** (p. 1981) operation on a derived class of the **DDSTypeSupport** (p. 1897) interface.

MT Safety:

UNSAFE. It is not safe to create a topic while another thread is trying to lookup that topic description with **DDSDomainParticipant::lookup_topicdescription** (p. 1658).

Parameters

topic_name	<<***in***>> *(p. 433)* Name for the new topic, must not exceed 255 characters. Cannot be NULL.
type_name	<<***in***>> *(p. 433)* The type to which the new **DDSTopic** (p. 1885) will be bound. Cannot be NULL.
library_name	<<***in***>> *(p. 433)* Library name containing the XML QoS profile. If library_name is null RTI Connext will use the default library (see **DDSDomainParticipant::set_default_library** (p. 1635)).
profile_name	<<***in***>> *(p. 433)* XML QoS Profile name. If profile_name is null RTI Connext will use the default profile (see **DDSDomainParticipant::set_default_profile** (p. 1636)).
listener	<<***in***>> *(p. 433).* Listener to be attached to the newly created **DDSTopic** (p. 1885).
mask	<<***in***>> *(p. 433).* Changes of communication status to be invoked on the listener. See **DDS_StatusMask** (p. 537).

Returns

newly created topic, or NULL on failure

See also

Specifying QoS on entities (p. 555) for information on setting QoS before entity creation

DDS_TopicQos (p. 1397) for rules on consistency among QoS

DDSDomainParticipant::create_topic (p. 1650)

DDSDomainParticipant::get_default_topic_qos (p. 1636)

DDSTopic::set_listener (p. 1889)

9.253.2.36 delete_topic()

```
virtual DDS_ReturnCode_t DDSDomainParticipant::delete_topic (
            DDSTopic * topic ) [pure virtual]
```

Deletes a **DDSTopic** (p. 1885).

Precondition

If the **DDSTopic** (p. 1885) does not belong to the application's **DDSDomainParticipant** (p. 1619), this operation fails with **DDS_RETCODE_PRECONDITION_NOT_MET** (p. 532).

Make sure no objects are using the topic. More specifically, there must be no existing **DDSDataReader** (p. 1556), **DDSDataWriter** (p. 1589), **DDSContentFilteredTopic** (p. 1551), or **DDSMultiTopic** (p. 1797) objects belonging to the same **DDSDomainParticipant** (p. 1619) that are using the **DDSTopic** (p. 1885). If delete_topic is called on a **DDSTopic** (p. 1885) with any of these existing objects attached to it, it will fail with **DDS_RETCODE_↩ PRECONDITION_NOT_MET** (p. 532).

Postcondition

Listener installed on the **DDSTopic** (p. 1885) will not be called after this method completes successfully.

MT Safety:

UNSAFE. It is not safe to delete an entity while another thread may be simultaneously calling an API that uses the entity.

Parameters

topic	<<***in***>> *(p. 433)* **DDSTopic** (p. 1885) to be deleted.

Exceptions

One	of the **Standard Return Codes** (p. 532), or **DDS_RETCODE_PRECONDITION_NOT_MET** (p. 532)

9.253.2.37 create_contentfilteredtopic()

```
virtual DDSContentFilteredTopic * DDSDomainParticipant::create_contentfilteredtopic (
            const char * name,
             DDSTopic * related_topic,
            const char * filter_expression,
            const  DDS_StringSeq & expression_parameters )  [pure virtual]
```

Creates a **DDSContentFilteredTopic** (p. 1551), that can be used to do content-based subscriptions.

The **DDSContentFilteredTopic** (p. 1551) only relates to samples published under that **DDSTopic** (p. 1885), filtered according to their content. The filtering is done by means of evaluating a logical expression that involves the values of some of the data-fields in the sample. The logical expression derived from the `filter_expression` and `expression_parameters` arguments.

Queries and Filters Syntax (p. 184) describes the syntax of `filter_expression` and `expression_parameters`.

Precondition

The application is not allowed to create two **DDSContentFilteredTopic** (p. 1551) objects with the same `topic_name` attached to the same **DDSDomainParticipant** (p. 1619). If the application attempts this, this method will fail and returns NULL.

If `related_topic` does not belong to this **DDSDomainParticipant** (p. 1619), this operation returns NULL.

This function will create a content filter using the builtin SQL filter which implements a superset of the DDS specification. This filter **requires** that all IDL types have been compiled with typecodes. If this precondition is not met, this operation returns NULL.

Parameters

name	<<***in***>> *(p. 433)* Name for the new content filtered topic, must not exceed 255 characters. Cannot be NULL.
related_topic	<<***in***>> *(p. 433)* **DDSTopic** (p. 1885) to be filtered. Cannot be NULL.
filter_expression	<<***in***>> *(p. 433)* Cannot be NULL
expression_parameters	<<***in***>> *(p. 433)* An empty sequence **must** be used if the filter expression does not contain any parameters. Length of sequence cannot be greater than 100.

Returns

newly created **DDSContentFilteredTopic** (p. 1551), or NULL on failure

9.253.2.38 create_contentfilteredtopic_with_filter()

```
virtual DDSContentFilteredTopic * DDSDomainParticipant::create_contentfilteredtopic_with_filter (
            const char * name,
```

```
 DDSTopic * related_topic,
const char * filter_expression,
const  DDS_StringSeq & expression_parameters,
const char * filter_name =  DDS_SQLFILTER_NAME )  [pure virtual]
```

<<***extension***>> *(p. 432)* Creates a **DDSContentFilteredTopic** (p. 1551) using the specified filter to do content-based subscriptions.

Parameters

name	<<***in***>> *(p. 433)* Name for the new content filtered topic. Cannot exceed 255 characters. Cannot be NULL.
related_topic	<<***in***>> *(p. 433)* **DDSTopic** (p. 1885) to be filtered. Cannot be NULL.
filter_expression	<<***in***>> *(p. 433)* Cannot be NULL.
expression_parameters	<<***in***>> *(p. 433)* . An empty sequence **must** be used if the filter expression does not contain any parameters. Length of the sequence cannot be greater than 100.
filter_name	<<***in***>> *(p. 433)* Name of content filter to use. Must previously have been registered with **DDSDomainParticipant::register_contentfilter** (p. 1632) on the same **DDSDomainParticipant** (p. 1619). Cannot be NULL. Builtin filter names are **DDS_SQLFILTER_NAME** (p. 61) and **DDS_STRINGMATCHFILTER_NAME** (p. 61)

Returns

newly created **DDSContentFilteredTopic** (p. 1551), or NULL on failure

9.253.2.39 delete_contentfilteredtopic()

```
virtual  DDS_ReturnCode_t DDSDomainParticipant::delete_contentfilteredtopic (
             DDSContentFilteredTopic * a_contentfilteredtopic )  [pure virtual]
```

Deletes a **DDSContentFilteredTopic** (p. 1551).

Precondition

The deletion of a **DDSContentFilteredTopic** (p. 1551) is not allowed if there are any existing **DDSData↩ Reader** (p. 1556) objects that are using the **DDSContentFilteredTopic** (p. 1551). If the operation is called on a **DDSContentFilteredTopic** (p. 1551) with existing **DDSDataReader** (p. 1556) objects attached to it, it will fail with **DDS_RETCODE_PRECONDITION_NOT_MET** (p. 532).

The **DDSContentFilteredTopic** (p. 1551) must be created by this **DDSDomainParticipant** (p. 1619), or else this operation will fail with **DDS_RETCODE_PRECONDITION_NOT_MET** (p. 532).

MT Safety:

UNSAFE. It is not safe to delete an entity while another thread may be simultaneously calling an API that uses the entity.

Parameters

a_contentfilteredtopic	<<***in***>> *(*p. *433)*

Exceptions

One	of the **Standard Return Codes** (p. 532) or **DDS_RETCODE_PRECONDITION_NOT_MET** (p. 532)

9.253.2.40 create_multitopic()

```
virtual  DDSMultiTopic * DDSDomainParticipant::create_multitopic (
             const char * name,
             const char * type_name,
             const char * subscription_expression,
             const  DDS_StringSeq & expression_parameters )  [pure virtual]
```

[Not supported (optional)] Creates a MultiTopic that can be used to subscribe to multiple topics and combine/filter the received data into a resulting type.

The resulting type is specified by the `type_name` argument. The list of topics and the logic used to combine, filter, and rearrange the information from each **DDSTopic** (p. 1885) are specified using the `subscription_expression` and `expression_parameters` arguments.

Queries and Filters Syntax (p. 184) describes the syntax of `subscription_expression` and `expression_parameters`.

Precondition

The application is not allowed to create two **DDSTopicDescription** (p. 1892) objects with the same `name` attached to the same **DDSDomainParticipant** (p. 1619). If the application attempts this, this method will fail and return NULL.

Prior to creating a **DDSMultiTopic** (p. 1797), the type must have been registered with RTI Connext. This is done using the **FooTypeSupport::register_type** (p. 1981) operation on a derived class of the **DDSTypeSupport** (p. 1897) interface. Otherwise, this method will return NULL.

Parameters

name	<<***in***>> *(*p. *433)* Name of the newly create **DDSMultiTopic** (p. 1797). Cannot be NULL.
type_name	<<***in***>> *(*p. *433)* Cannot be NULL.
subscription_expression	<<***in***>> *(*p. *433)* Cannot be NULL.
expression_parameters	<<***in***>> *(*p. *433)*

Returns

NULL

9.253.2.41 delete_multitopic()

```
virtual  DDS_ReturnCode_t DDSDomainParticipant::delete_multitopic (
             DDSMultiTopic * a_multitopic )  [pure virtual]
```

[Not supported (optional)] Deletes a **DDSMultiTopic** (p. 1797).

Precondition

The deletion of a **DDSMultiTopic** (p. 1797) is not allowed if there are any existing **DDSDataReader** (p. 1556) objects that are using the **DDSMultiTopic** (p. 1797). If the delete_multitopic operation is called on a **DDSMultiTopic** (p. 1797) with existing **DDSDataReader** (p. 1556) objects attached to it, it will fail with **DDS_RETCODE_PRECONDITION_NOT_MET** (p. 532).

The **DDSMultiTopic** (p. 1797) must be created by this **DDSDomainParticipant** (p. 1619), or else this operation will fail with **DDS_RETCODE_PRECONDITION_NOT_MET** (p. 532).

MT Safety:

UNSAFE. It is not safe to delete an entity while another thread may be simultaneously calling an API that uses the entity.

Parameters

a_multitopic	<<***in***>> (p. *433*)

9.253.2.42 find_topic()

```
virtual  DDSTopic * DDSDomainParticipant::find_topic (
             const char * topic_name,
             const  DDS_Duration_t & timeout )  [pure virtual]
```

Finds an existing (or ready to exist) **DDSTopic** (p. 1885), based on its name.

This call can be used to block for a specified duration to wait for the **DDSTopic** (p. 1885) to be created.

If the requested **DDSTopic** (p. 1885) already exists, it is returned. Otherwise, **find_topic()** (p. 1657) waits until another thread creates it or else returns when the specified timeout occurs.

find_topic() (p. 1657) is useful when multiple threads are concurrently creating and looking up topics. In that case, one thread can call **find_topic()** (p. 1657) and, if another thread has not yet created the topic being looked up, it can wait for some period of time for it to do so. In almost all other cases, it is more straightforward to call **DDSDomainParticipant::lookup_topicdescription** (p. 1658).

The **DDSDomainParticipant** (p. 1619) must already be enabled.

Note: *Each* **DDSTopic** (p. 1885) obtained by **DDSDomainParticipant::find_topic** (p. 1657) must also be deleted by means of **DDSDomainParticipant::delete_topic** (p. 1653). If **DDSTopic** (p. 1885) is obtained multiple times by means of **DDSDomainParticipant::find_topic** (p. 1657) or **DDSDomainParticipant::create_topic** (p. 1650), it must also be deleted that same number of times using **DDSDomainParticipant::delete_topic** (p. 1653).

Parameters

topic_name	<<***in***>> *(p. 433)* Name of the **DDSTopic** (p. 1885) to search for. Cannot be NULL.
timeout	<<***in***>> *(p. 433)* The time to wait if the **DDSTopic** (p. 1885) does not exist already.

Returns

the topic, if it exists, or NULL

9.253.2.43 lookup_topicdescription()

```
virtual DDSTopicDescription * DDSDomainParticipant::lookup_topicdescription (
            const char * topic_name ) [pure virtual]
```

Looks up an existing, locally created **DDSTopicDescription** (p. 1892), based on its name.

DDSTopicDescription (p. 1892) is the base class for **DDSTopic** (p. 1885), **DDSMultiTopic** (p. 1797) and **DDSContentFilteredTopic** (p. 1551). So you can narrow the **DDSTopicDescription** (p. 1892) returned from this operation to a **DDSTopic** (p. 1885) or **DDSContentFilteredTopic** (p. 1551) as appropriate.

Unlike **DDSDomainParticipant::find_topic** (p. 1657), which logically returns a new **DDSTopic** (p. 1885) object that must be independently deleted, *this* operation returns a reference to the original local object.

The **DDSDomainParticipant** (p. 1619) does not have to be enabled when you call **lookup_topicdescription()** (p. 1658).

The returned topic may be either enabled or disabled.

MT Safety:

UNSAFE. It is not safe to lookup a topic description while another thread is creating that topic.

Parameters

topic_name	<<***in***>> *(p. 433)* Name of **DDSTopicDescription** (p. 1892) to search for. This string must be no more than 255 characters; it cannot be NULL.

Returns

The topic description, if it has already been created locally, otherwise it returns NULL.

9.253.2.44 create_flowcontroller()

```
virtual  DDSFlowController * DDSDomainParticipant::create_flowcontroller (
            const char * name,
            const  DDS_FlowControllerProperty_t & prop )  [pure virtual]
```

<<***extension***>> *(p. 432)* Creates a **DDSFlowController** (p. 1736) with the desired property.

The created **DDSFlowController** (p. 1736) is associated with a **DDSDataWriter** (p. 1589) via **DDS_PublishModeQos↩ Policy::flow_controller_name** (p. 1281). A single FlowController may service multiple DataWriters instances, even if they belong to a different **DDSPublisher** (p. 1818). The `property` determines how the FlowController shapes the network traffic.

Precondition

The specified `property` must be consistent, or the operation will fail and no **DDSFlowController** (p. 1736) will be created.

MT Safety:

UNSAFE. If **DDS_FLOW_CONTROLLER_PROPERTY_DEFAULT** (p. 60) is used for `property`, it is not safe to create the flow controller while another thread may be simultaneously calling **DDSDomainParticipant::set_↩ default_flowcontroller_property** (p. 1631) or trying to lookup that flow controller with **DDSDomainParticipant↩ ::lookup_flowcontroller** (p. 1660).

Parameters

name	<<***in***>> *(p. 433)* name of the **DDSFlowController** (p. 1736) to create. A **DDSDataWriter** (p. 1589) is associated with a **DDSFlowController** (p. 1736) by name. Limited to 255 characters.
prop	<<***in***>> *(p. 433)* property to be used for creating the new **DDSFlowController** (p. 1736). The special value **DDS_FLOW_CONTROLLER_PROPERTY_DEFAULT** (p. 60) can be used to indicate that the **DDSFlowController** (p. 1736) should be created with the default **DDS_FlowControllerProperty_t** (p. 1156) set in the **DDSDomainParticipant** (p. 1619).

Returns

Newly created flow controller object or NULL on failure.

See also

DDS_FlowControllerProperty_t (p. 1156) for rules on consistency among property

DDS_FLOW_CONTROLLER_PROPERTY_DEFAULT (p. 60)

DDSDomainParticipant::get_default_flowcontroller_property (p. 1630)

9.253.2.45 delete_flowcontroller()

```
virtual DDS_ReturnCode_t DDSDomainParticipant::delete_flowcontroller (
            DDSFlowController * fc ) [pure virtual]
```

<<***extension***>> *(p. 432)* Deletes an existing **DDSFlowController** (p. 1736).

Precondition

The **DDSFlowController** (p. 1736) must not have any attached **DDSDataWriter** (p. 1589) objects. If there are any attached **DDSDataWriter** (p. 1589) objects, it will fail with **DDS_RETCODE_PRECONDITION_NOT_MET** (p. 532).

The **DDSFlowController** (p. 1736) must have been created by this **DDSDomainParticipant** (p. 1619), or else it will fail with **DDS_RETCODE_PRECONDITION_NOT_MET** (p. 532).

Postcondition

The **DDSFlowController** (p. 1736) is deleted if this method completes successfully.

MT Safety:

UNSAFE. It is not safe to delete an entity while another thread may be simultaneously calling an API that uses the entity.

Parameters

fc	<<***in***>> *(p. 433)* The **DDSFlowController** (p. 1736) to be deleted.

Exceptions

One	of the **Standard Return Codes** (p. 532), or **DDS_RETCODE_PRECONDITION_NOT_MET** (p. 532).

9.253.2.46 lookup_flowcontroller()

```
virtual DDSFlowController * DDSDomainParticipant::lookup_flowcontroller (
            const char * name ) [pure virtual]
```

<<***extension***>> *(p. 432)* Looks up an existing locally-created **DDSFlowController** (p. 1736), based on its name.

Looks up a previously created **DDSFlowController** (p. 1736), including the built-in ones. Once a **DDSFlowController** (p. 1736) has been deleted, subsequent lookups will fail.

MT Safety:

UNSAFE. It is not safe to lookup a flow controller description while another thread is creating that flow controller.

Parameters

name	<<***in***>> *(p. 433)* Name of **DDSFlowController** (p. 1736) to search for. Limited to 255 characters. Cannot be NULL.

Returns

The flow controller if it has already been created locally, or NULL otherwise.

9.253.2.47 get_builtin_subscriber()

```
virtual DDSSubscriber * DDSDomainParticipant::get_builtin_subscriber ( ) [pure virtual]
```

Accesses the **built-in DDSSubscriber** (p. 1860).

Each **DDSDomainParticipant** (p. 1619) contains several built-in **DDSTopic** (p. 1885) objects as well as corresponding **DDSDataReader** (p. 1556) objects to access them. All of these **DDSDataReader** (p. 1556) objects belong to a single built-in **DDSSubscriber** (p. 1860).

The built-in Topics are used to communicate information about other **DDSDomainParticipant** (p. 1619), **DDSTopic** (p. 1885), **DDSDataReader** (p. 1556), and **DDSDataWriter** (p. 1589) objects.

The built-in subscriber is created when this operation is called for the first time. The built-in subscriber is deleted automatically when the **DDSDomainParticipant** (p. 1619) is deleted.

Returns

The built-in **DDSSubscriber** (p. 1860) singleton.

See also

DDS_SubscriptionBuiltinTopicData (p. 1370)

DDS_PublicationBuiltinTopicData (p. 1264)

DDS_ParticipantBuiltinTopicData (p. 1229)

DDS_TopicBuiltinTopicData (p. 1391)

9.253.2.48 ignore_participant()

```
virtual DDS_ReturnCode_t DDSDomainParticipant::ignore_participant (
            const DDS_InstanceHandle_t & handle ) [pure virtual]
```

Instructs RTI Connext to locally ignore a remote **DDSDomainParticipant** (p. 1619).

From the time of this call onwards, RTI Connext will locally behave as if the remote participant did not exist. This means it will ignore any topic, publication, or subscription that originates on that **DDSDomainParticipant** (p. 1619).

There is no way to reverse this operation.

This operation can be used in conjunction with the discovery of remote participants offered by means of the **DDS_↩ ParticipantBuiltinTopicData** (p. 1229) to provide access control.

Application data can be associated with a **DDSDomainParticipant** (p. 1619) by means of the **USER_DATA** (p. 653) policy. This application data is propagated as a field in the built-in topic and can be used by an application to implement its own access control policy.

The **DDSDomainParticipant** (p. 1619) to ignore is identified by the handle argument. This `handle` is the one that appears in the **DDS_SampleInfo** (p. 1344) retrieved when reading the data-samples available for the built-in **DDSData↩ Reader** (p. 1556) to the **DDSDomainParticipant** (p. 1619) topic. The built-in **DDSDataReader** (p. 1556) is read with the same **FooDataReader::read** (p. 1919) and **FooDataReader::take** (p. 1920) operations used for any **DDSDataReader** (p. 1556).

Parameters

handle	*<<**in**>> (p. 433)* **DDS_InstanceHandle_t** (p. 76) of the **DDSDomainParticipant** (p. 1619) to be ignored.

Exceptions

One	of the **Standard Return Codes** (p. 532), **DDS_RETCODE_OUT_OF_RESOURCES** (p. 533), **DDS_RETCODE_NOT_ENABLED** (p. 533)

See also

DDS_ParticipantBuiltinTopicData (p. 1229)

DDS_PARTICIPANT_TOPIC_NAME (p. 491)

DDSDomainParticipant::get_builtin_subscriber (p. 1661)

9.253.2.49 banish_ignored_participants()

```
virtual DDS_ReturnCode_t DDSDomainParticipant::banish_ignored_participants ( ) [pure virtual]
```

*<<**extension**>> (p. 432)* Prevents ignored remote DomainParticipants from receiving traffic from the local **DDSDomainParticipant** (p. 1619).

This method complements **DDSDomainParticipant::ignore_participant** (p. 1661): ignore_participant prevents the local **DDSDomainParticipant** (p. 1619) from processing traffic from the remote DomainParticipant, while this method prevents already ignored remote DomainParticipants from processing traffic from the local DomainParticipant.

Note: this method is currently only supported when enabling the RTI Security Plugins. Please refer to the `RTI Security Plugins User's Manual` for more information.

MT Safety:

Safe.

Exceptions

One	of the **Standard Return Codes** (p. 532), **DDS_RETCODE_PRECONDITION_NOT_MET** (p. 532), **DDS_RETCODE_NOT_ENABLED** (p. 533)

See also

DDSDomainParticipant::ignore_participant (p. 1661)

9.253.2.50 ignore_topic()

```
virtual DDS_ReturnCode_t DDSDomainParticipant::ignore_topic (
            const DDS_InstanceHandle_t & handle ) [pure virtual]
```

Instructs RTI Connext to locally ignore a **DDSTopic** (p. 1885).

This means it will locally ignore any publication, or subscription to the **DDSTopic** (p. 1885).

There is no way to reverse this operation.

This operation can be used to save local resources when the application knows that it will never publish or subscribe to data under certain topics.

The **DDSTopic** (p. 1885) to ignore is identified by the `handle` argument. This is the handle of a **DDSTopic** (p. 1885) that appears in the **DDS_SampleInfo** (p. 1344) retrieved when reading data samples from the built-in **DDSDataReader** (p. 1556) for the **DDSTopic** (p. 1885).

Parameters

handle	<<***in***>> *(p. 433)* Handle of the **DDSTopic** (p. 1885) to be ignored.

Exceptions

One	of the **Standard Return Codes** (p. 532), **DDS_RETCODE_OUT_OF_RESOURCES** (p. 533) or **DDS_RETCODE_NOT_ENABLED** (p. 533)

See also

DDS_TopicBuiltinTopicData (p. 1391)

DDS_TOPIC_TOPIC_NAME (p. 493)

DDSDomainParticipant::get_builtin_subscriber (p. 1661)

9.253.2.51 ignore_publication()

```
virtual DDS_ReturnCode_t DDSDomainParticipant::ignore_publication (
            const DDS_InstanceHandle_t & handle ) [pure virtual]
```

Instructs RTI Connext to locally ignore a publication.

A publication is defined by the association of a topic name, user data, and partition set on the **DDSPublisher** (p. 1818) (see **DDS_PublicationBuiltinTopicData** (p. 1264)). After this call, any data written by that publication's **DDSDataWriter** (p. 1589) will be ignored.

This operation can be used to ignore local *and* remote DataWriters.

The publication (DataWriter) to ignore is identified by the `handle` argument.

- To ignore a *remote* DataWriter, the `handle` can be obtained from the **DDS_SampleInfo** (p. 1344) retrieved when reading data samples from the built-in **DDSDataReader** (p. 1556) for the publication topic.

- To ignore a *local* DataWriter, the `handle` can be obtained by calling **DDSEntity::get_instance_handle** (p. 1736) for the local DataWriter.

There is no way to reverse this operation.

Parameters

handle	<<***in***>> *(p. 433)* Handle of the **DDSDataWriter** (p. 1589) to be ignored.

Exceptions

One	of the **Standard Return Codes** (p. 532), **DDS_RETCODE_OUT_OF_RESOURCES** (p. 533) or **DDS_RETCODE_NOT_ENABLED** (p. 533)

See also

DDS_PublicationBuiltinTopicData (p. 1264)

DDS_PUBLICATION_TOPIC_NAME (p. 494)

DDSDomainParticipant::get_builtin_subscriber (p. 1661)

9.253.2.52 ignore_subscription()

```
virtual DDS_ReturnCode_t DDSDomainParticipant::ignore_subscription (
            const DDS_InstanceHandle_t & handle ) [pure virtual]
```

Instructs RTI Connext to locally ignore a subscription.

A subscription is defined by the association of a topic name, user data, and partition set on the **DDSSubscriber** (p. 1860) (see **DDS_SubscriptionBuiltinTopicData** (p. 1370)). After this call, any data received related to that subscription's **DDSDataReader** (p. 1556) will be ignored.

This operation can be used to ignore local *and* remote DataReaders.

The subscription to ignore is identified by the `handle` argument.

- To ignore a *remote* DataReader, the `handle` can be obtained from the **DDS_SampleInfo** (p. 1344) retrieved when reading data samples from the built-in **DDSDataReader** (p. 1556) for the subscription topic.

- To ignore a *local* DataReader, the `handle` can be obtained by calling **DDSEntity::get_instance_handle** (p. 1736) for the local DataReader.

There is no way to reverse this operation.

Parameters

handle	*<<**in**>> (p. 433)* Handle of the **DDSDataReader** (p. 1556) to be ignored.

Exceptions

One	of the **Standard Return Codes** (p. 532), **DDS_RETCODE_OUT_OF_RESOURCES** (p. 533) or **DDS_RETCODE_NOT_ENABLED** (p. 533)

See also

DDS_SubscriptionBuiltinTopicData (p. 1370)

DDS_SUBSCRIPTION_TOPIC_NAME (p. 496)

DDSDomainParticipant::get_builtin_subscriber (p. 1661)

9.253.2.53 get_domain_id()

```
virtual DDS_DomainId_t DDSDomainParticipant::get_domain_id ( ) [pure virtual]
```

Get the unique domain identifier.

This operation retrieves the domain id used to create the **DDSDomainParticipant** (p. 1619). The domain id identifies the DDS domain to which the **DDSDomainParticipant** (p. 1619) belongs. Each DDS domain represents a separate data 'communication plane' isolated from other domains.

Returns

the unique `domainId` that was used to create the domain

See also

DDSDomainParticipantFactory::create_participant (p. 1710)

DDSDomainParticipantFactory::create_participant_with_profile (p. 1711)

9.253.2.54 get_current_time()

```
virtual DDS_ReturnCode_t DDSDomainParticipant::get_current_time (
            DDS_Time_t & current_time ) [pure virtual]
```

Returns the current value of the time.

The current value of the time that RTI Connext uses to time-stamp **DDSDataWriter** (p. 1589) and to set the reception-timestamp for the data updates that it receives.

Parameters

current_time	<<***inout***>> *(p. 433)* Current time to be filled up.

Exceptions

One	of the **Standard Return Codes** (p. 532)

9.253.2.55 register_durable_subscription()

```
virtual DDS_ReturnCode_t DDSDomainParticipant::register_durable_subscription (
            const DDS_EndpointGroup_t & group,
            const char * topic_name ) [pure virtual]
```

<<***extension***>> *(p. 432)* Registers a Durable Subscription on the specified **DDSTopic** (p. 1885) on all Persistence Services.

If you need to receive all samples published on a **DDSTopic** (p. 1885), including the ones published while a **DDSDataReader** (p. 1556) is inactive or before it may be created, create a Durable Subscription using this method.

In this way, the Persistence Service will ensure that all the samples on that **DDSTopic** (p. 1885) are retained until they are acknowledged by at least *N* DataReaders belonging to the Durable Subscription where *N* is the quorum count.

If the same Durable Subscription is created on a different **DDSTopic** (p. 1885), the Persistence Service will implicitly delete the previous Durable Subscription and create a new one on the new **DDSTopic** (p. 1885).

Parameters

group	<<***in***>> *(p. 433)* **DDS_EndpointGroup_t** (p. 1142) The Durable Subscription name and quorum.
topic_name	<<***in***>> *(p. 433)* The topic name for which the Durable Subscription is created.

Exceptions

One	of the **Standard Return Codes** (p. 532)

9.253.2.56 delete_durable_subscription()

```
virtual DDS_ReturnCode_t DDSDomainParticipant::delete_durable_subscription (
            const DDS_EndpointGroup_t & group ) [pure virtual]
```

<<***extension***>> *(p. 432)* Deletes an existing Durable Subscription on all Persistence Services.

The Persistence Service will delete the Durable Subscription and the quorum of the existing samples will be considered satisfied.

Parameters

group	<<***in***>> *(p. 433)* **DDS_EndpointGroup_t** (p. 1142) specifying the Durable Subscription name. Quorum is not required for this operation.

Exceptions

One	of the **Standard Return Codes** (p. 532)

9.253.2.57 assert_liveliness()

```
virtual DDS_ReturnCode_t DDSDomainParticipant::assert_liveliness ( ) [pure virtual]
```

Manually asserts the liveliness of this **DDSDomainParticipant** (p. 1619).

This is used in combination with the **DDS_LivelinessQosPolicy** (p. 1180) to indicate to RTI Connext that the entity remains active.

You need to use this operation if the **DDSDomainParticipant** (p. 1619) contains **DDSDataWriter** (p. 1589) entities with the **DDS_LivelinessQosPolicy::kind** (p. 1182) set to **DDS_MANUAL_BY_PARTICIPANT_LIVELINESS_QOS** (p. 608) and it only affects the liveliness of those **DDSDataWriter** (p. 1589) entities. Otherwise, it has no effect.

Note: writing data via the **FooDataWriter::write** (p. 1951) or **FooDataWriter::write_w_timestamp** (p. 1955) operation asserts liveliness on the **DDSDataWriter** (p. 1589) itself and its **DDSDomainParticipant** (p. 1619). Consequently the use of **assert_liveliness()** (p. 1667) is only needed if the application is not writing data regularly.

Exceptions

One	of the **Standard Return Codes** (p. 532), or **DDS_RETCODE_NOT_ENABLED** (p. 533)

See also

DDS_LivelinessQosPolicy (p. 1180)

9.253.2.58 resume_endpoint_discovery()

```
virtual  DDS_ReturnCode_t DDSDomainParticipant::resume_endpoint_discovery (
            const  DDS_InstanceHandle_t & remote_participant_handle )  [pure virtual]
```

<<***extension***>> *(p. 432)* Initiates endpoint discovery with the specified remote **DDSDomainParticipant** (p. 1619).

If the operation returns **DDS_RETCODE_OK** (p. 532), the **DDSDomainParticipant** (p. 1619) will initiate endpoint discovery with the remote **DDSDomainParticipant** (p. 1619) provided as a parameter.

When **DDS_DiscoveryQosPolicy::enable_endpoint_discovery** (p. 964) is set to **DDS_BOOLEAN_FALSE** (p. 513), this operation allows the RTI Connext application to select for which remote DomainParticipants endpoint discovery is performed. By disabling endpoint discovery, the DomainParticipant will not store any state about remote endpoints and will not send local endpoint information to remote DomainParticipants.

If **DDS_DiscoveryQosPolicy::enable_endpoint_discovery** (p. 964) is set to **DDS_BOOLEAN_TRUE** (p. 513), endpoint discovery will automatically occur for every discovered **DDSDomainParticipant** (p. 1619). In this case, invoking this operation will have no effect and will return **DDS_RETCODE_OK** (p. 532).

When **DDS_DiscoveryQosPolicy::enable_endpoint_discovery** (p. 964) is set to **DDS_BOOLEAN_FALSE** (p. 513), you have two options after a remote **DDSDomainParticipant** (p. 1619) is discovered:

- Call this operation to enable endpoint discovery. After invoking this operation, the **DDSDomainParticipant** (p. 1619) will start to exchange endpoint information so that matching and communication can occur with the remote **DDSDomainParticipant** (p. 1619).
- Call the **DDSDomainParticipant::ignore_participant** (p. 1661) operation to permanently ignore endpoint discovery with the remote **DDSDomainParticipant** (p. 1619).

Setting **DDS_DiscoveryQosPolicy::enable_endpoint_discovery** (p. 964) to **DDS_BOOLEAN_FALSE** (p. 513) enables application-level authentication use cases, in which a **DDSDomainParticipant** (p. 1619) will initiate endpoint discovery with a remote **DDSDomainParticipant** (p. 1619) after successful authentication at the application level.

The `remote_participant_handle` paremeter is the one that appears in the **DDS_SampleInfo** (p. 1344) retrieved when reading the data samples available for the built-in **DDSParticipantBuiltinTopicDataDataReader** (p. 1816).

If the specified remote **DDSDomainParticipant** (p. 1619) is not in the database of discovered DomainParticipants or has been previously ignored, this operation will fail with **DDS_RETCODE_ERROR** (p. 532).

This operation can be called multiple times on the same remote participant. If endpoint discovery has already been resumed, successive calls will have no effect and will return **DDS_RETCODE_OK** (p. 532).

Parameters

remote_participant_handle	*<<**in**>> (p. 433)* Handle of a discovered **DDSDomainParticipant** (p. 1619) for which endpoint discovery is to be resumed.

Exceptions

One	of the **Standard Return Codes** (p. 532),or **DDS_RETCODE_NOT_ENABLED** (p. 533)

See also

DDS_DiscoveryQosPolicy (p. 960)

9.253.2.59 delete_contained_entities()

```
virtual DDS_ReturnCode_t DDSDomainParticipant::delete_contained_entities ( ) [pure virtual]
```

Delete all the entities that were created by means of the "create" operations on the **DDSDomainParticipant** (p. 1619).

This operation deletes all contained **DDSPublisher** (p. 1818) (including an implicit Publisher, if one exists), **DDSSubscriber** (p. 1860) (including implicit Subscriber), **DDSTopic** (p. 1885), **DDSContentFilteredTopic** (p. 1551), and **DDSMultiTopic** (p. 1797) objects.

Prior to deleting each contained entity, this operation will recursively call the corresponding **delete_contained_entities()** (p. 1669) operation on each contained entity (if applicable). This pattern is applied recursively. In this manner the operation **delete_contained_entities()** (p. 1669) on the **DDSDomainParticipant** (p. 1619) will end up recursively deleting all the entities contained in the **DDSDomainParticipant** (p. 1619), including the **DDSDataWriter** (p. 1589), **DDSDataReader** (p. 1556), as well as the **DDSQueryCondition** (p. 1841), **DDSReadCondition** (p. 1842), and **DDSTopicQuery** (p. 1895) objects belonging to the contained **DDSDataReader** (p. 1556).

The operation will fail with **DDS_RETCODE_PRECONDITION_NOT_MET** (p. 532) if any of the contained entities is in a state where it cannot be deleted .

If **delete_contained_entities()** (p. 1669) completes successfully, the application may delete the **DDSDomainParticipant** (p. 1619).

MT Safety:

UNSAFE. It is not safe to delete an entity while another thread may be simultaneously calling an API that uses the entity.

Exceptions

One	of the **Standard Return Codes** (p. 532), or **DDS_RETCODE_PRECONDITION_NOT_MET** (p. 532).

Examples

HelloWorld_publisher.cxx, and **HelloWorld_subscriber.cxx**.

9.253.2.60 get_discovered_participants()

```
virtual DDS_ReturnCode_t DDSDomainParticipant::get_discovered_participants (
            DDS_InstanceHandleSeq & participant_handles ) [pure virtual]
```

Returns a list of discovered **DDSDomainParticipant** (p. 1619) entities.

This operation retrieves the list of **DDSDomainParticipant** (p. 1619) entities that have been discovered in the domain and that the application has not indicated should be "ignored" by means of the **DDSDomainParticipant::ignore_participant** (p. 1661) operation. When using **DDS_DISCOVERYCONFIG_BUILTIN_SPDP2** (p. 592), this list only includes **DDSDomainParticipant** (p. 1619) entities that the application has received configuration information from.

Parameters

participant_handles	<<***inout***>> *(p. 433)* **DDS_InstanceHandleSeq** (p. 1166) to be filled with handles of the discovered **DDSDomainParticipant** (p. 1619) entities.

Exceptions

One	of the **Standard Return Codes** (p. 532) or **DDS_RETCODE_NOT_ENABLED** (p. 533)

9.253.2.61 get_discovered_participants_from_subject_name()

```
virtual DDS_ReturnCode_t DDSDomainParticipant::get_discovered_participants_from_subject_name (
            DDS_InstanceHandleSeq & participant_handles,
           const char * subject_name ) [pure virtual]
```

<<***extension***>> *(p. 432)* Returns a list of discovered **DDSDomainParticipant** (p. 1619) entities that have the given **DDS_EntityNameQosPolicy::name** (p. 1148).

This operation retrieves the same list as **DDSDomainParticipant::get_discovered_participants** (p. 1670), except this list contains only the participants that have the given **DDS_EntityNameQosPolicy::name** (p. 1148).

Note: this method has different functionality when enabling the RTI Security Plugins. Please refer to the `RTI Security Plugins User's Manual` for more information.

MT Safety:

Safe.

Parameters

participant_handles	*<<**inout**>> (p. 433)* **DDS_InstanceHandleSeq** (p. 1166) to be filled with handles of the discovered **DDSDomainParticipant** (p. 1619) entities.
subject_name	*<<**in**>> (p. 433)* The **DDS_EntityNameQosPolicy::name** (p. 1148) by which to filter the list.

Exceptions

One	of the **Standard Return Codes** (p. 532) or **DDS_RETCODE_NOT_ENABLED** (p. 533)

9.253.2.62 get_discovered_participant_data()

```
virtual DDS_ReturnCode_t DDSDomainParticipant::get_discovered_participant_data (
            struct DDS_ParticipantBuiltinTopicData & participant_data,
            const DDS_InstanceHandle_t & participant_handle ) [pure virtual]
```

Returns **DDS_ParticipantBuiltinTopicData** (p. 1229) for the specified **DDSDomainParticipant** (p. 1619).

This operation retrieves information on a **DDSDomainParticipant** (p. 1619) that has been discovered on the network. The participant must be in the same domain as the participant on which this operation is invoked and must not have been "ignored" by means of the **DDSDomainParticipant::ignore_participant** (p. 1661) operation.

The `participant_handle` must correspond to such a DomainParticipant. Otherwise, the operation will fail with **DDS_RETCODE_PRECONDITION_NOT_MET** (p. 532).

Use the operation **DDSDomainParticipant::get_discovered_participants** (p. 1670) to find the **DDSDomain↩ Participant** (p. 1619) entities that are currently discovered.

MT Safety:

Safe.

Parameters

participant_data	*<<**inout**>> (p. 433)* **DDS_ParticipantBuiltinTopicData** (p. 1229) to be filled in with the data for the specified **DDSDomainParticipant** (p. 1619).
participant_handle	*<<**in**>> (p. 433)* **DDS_InstanceHandle_t** (p. 76) of **DDSDomainParticipant** (p. 1619).

Exceptions

One	of the **Standard Return Codes** (p. 532), **DDS_RETCODE_PRECONDITION_NOT_MET** (p. 532) or **DDS_RETCODE_NOT_ENABLED** (p. 533)

See also

DDS_ParticipantBuiltinTopicData (p. 1229)

DDSDomainParticipant::get_discovered_participants (p. 1670)

9.253.2.63 get_discovered_participant_subject_name()

```
virtual DDS_ReturnCode_t DDSDomainParticipant::get_discovered_participant_subject_name (
            char * subject_name,
             DDS_UnsignedLong & subject_name_size,
            const DDS_InstanceHandle_t & participant_handle ) [pure virtual]
```

<<***extension***>> *(p. 432)* Returns **DDS_EntityNameQosPolicy::name** (p. 1148) for the specified **DDSDomain↩ Participant** (p. 1619).

This operation retrieves the **DDS_EntityNameQosPolicy::name** (p. 1148) of a **DDSDomainParticipant** (p. 1619) that has been discovered on the network. The participant must be in the same domain as the participant on which this operation is invoked and must not have been "ignored" by means of the **DDSDomainParticipant::ignore_participant** (p. 1661) operation.

The `participant_handle` must correspond to such a DomainParticipant. If the `participant_handle` is **DDS_HANDLE_NIL** (p. 78) or is not a valid **DDS_InstanceHandle_t** (p. 76) for a DomainParticipant, then the operation will fail with **DDS_RETCODE_BAD_PARAMETER** (p. 532). If the `participant_handle` corresponds to a DomainParticipant that has not been discovered, then the operation will fail with **DDS_RETCODE_PRECONDITION↩ _NOT_MET** (p. 532).

Use the operation **DDSDomainParticipant::get_discovered_participants** (p. 1670) to find the **DDSDomain↩ Participant** (p. 1619) entities that are currently discovered.

Note: this method has different functionality when enabling the RTI Security Plugins. Please refer to the `RTI Security Plugins User's Manual` for more information.

MT Safety:

Safe.

Parameters

subject_name	<<***out***>> *(p. 433)* The char buffer that will be used to store the **DDS_EntityNameQosPolicy::name** (p. 1148) of the discovered **DDSDomainParticipant** (p. 1619). If NULL, this function will return the required length of this buffer through the subject_name_size parameter.
subject_name_size	<<***inout***>> *(p. 433)* Cannot be NULL. The size of the supplied char buffer. If the supplied buffer is NULL, or if the supplied buffer is not large enough, this value will be updated with the required length of the buffer, which includes space for the NUL terminator. If the supplied buffer is not NULL but this value is not large enough, then this function will fail with **DDS_RETCODE_BAD_PARAMETER** (p. 532). If the **DDS_EntityNameQosPolicy::name** (p. 1148) is NULL, then this value will be set to 0.
participant_handle	<<***in***>> *(p. 433)* **DDS_InstanceHandle_t** (p. 76) of **DDSDomainParticipant** (p. 1619).

Exceptions

One	of the **Standard Return Codes** (p. 532), **DDS_RETCODE_PRECONDITION_NOT_MET** (p. 532) or **DDS_RETCODE_NOT_ENABLED** (p. 533)

See also

DDS_EntityNameQosPolicy::name (p. 1148)

DDSDomainParticipant::get_discovered_participants (p. 1670)

9.253.2.64 get_discovered_topics()

```
virtual  DDS_ReturnCode_t DDSDomainParticipant::get_discovered_topics (
             DDS_InstanceHandleSeq & topic_handles )  [pure virtual]
```

Returns list of discovered **DDSTopic** (p. 1885) objects.

This operation retrieves the list of **DDSTopic** (p. 1885) s that have been discovered in the domain and that the application has not indicated should be "ignored" by means of the **DDSDomainParticipant::ignore_topic** (p. 1663) operation.

Parameters

topic_handles	<<***inout***>> *(p. 433)* **DDS_InstanceHandleSeq** (p. 1166) to be filled with handles of the discovered **DDSTopic** (p. 1885) objects

Exceptions

One	of the **Standard Return Codes** (p. 532) or **DDS_RETCODE_NOT_ENABLED** (p. 533)

9.253.2.65 get_discovered_topic_data()

```
virtual  DDS_ReturnCode_t DDSDomainParticipant::get_discovered_topic_data (
             struct  DDS_TopicBuiltinTopicData & topic_data,
             const  DDS_InstanceHandle_t & topic_handle )  [pure virtual]
```

Returns **DDS_TopicBuiltinTopicData** (p. 1391) for the specified **DDSTopic** (p. 1885).

This operation retrieves information on a **DDSTopic** (p. 1885) that has been discovered by the local Participant and must not have been "ignored" by means of the **DDSDomainParticipant::ignore_topic** (p. 1663) operation.

The `topic_handle` must correspond to such a topic. Otherwise, the operation will fail with **DDS_RETCODE_↩ PRECONDITION_NOT_MET** (p. 532).

This call is not supported for remote topics. If a remote `topic_handle` is used, the operation will fail with **DDS_↩ RETCODE_UNSUPPORTED** (p. 532).

Use the operation **DDSDomainParticipant::get_discovered_topics** (p. 1673) to find the topics that are currently discovered.

Parameters

topic_data	*<<**inout**>> (p. 433)* **DDS_TopicBuiltinTopicData** (p. 1391) to be filled with the specified **DDSTopic** (p. 1885)'s data.
topic_handle	*<<**in**>> (p. 433)* **DDS_InstanceHandle_t** (p. 76) of **DDSTopic** (p. 1885).

Exceptions

One	of the **Standard Return Codes** (p. 532), **DDS_RETCODE_PRECONDITION_NOT_MET** (p. 532) or **DDS_RETCODE_NOT_ENABLED** (p. 533)

See also

DDS_TopicBuiltinTopicData (p. 1391)

DDSDomainParticipant::get_discovered_topics (p. 1673)

9.253.2.66 contains_entity()

```
virtual DDS_Boolean DDSDomainParticipant::contains_entity (
            const DDS_InstanceHandle_t & a_handle ) [pure virtual]
```

Completes successfully with **DDS_BOOLEAN_TRUE** (p. 513) if the referenced **DDSEntity** (p. 1731) is contained by the **DDSDomainParticipant** (p. 1619).

This operation checks whether or not the given `a_handle` represents an **DDSEntity** (p. 1731) that was created from the **DDSDomainParticipant** (p. 1619). The containment applies recursively. That is, it applies both to entities (**DDSTopicDescription** (p. 1892), **DDSPublisher** (p. 1818), or **DDSSubscriber** (p. 1860)) created directly using the **DDSDomainParticipant** (p. 1619) as well as entities created using a contained **DDSPublisher** (p. 1818), or **DDSSubscriber** (p. 1860) as the factory, and so forth.

The `instance` handle for an **DDSEntity** (p. 1731) may be obtained from built-in topic data, from various statuses, or from the operation **DDSEntity::get_instance_handle** (p. 1736).

Parameters

a_handle	*<<**in**>> (p. 433)* **DDS_InstanceHandle_t** (p. 76) of the **DDSEntity** (p. 1731) to be checked.

Returns

DDS_BOOLEAN_TRUE (p. 513) if **DDSEntity** (p. 1731) is contained by the **DDSDomainParticipant** (p. 1619), or **DDS_BOOLEAN_FALSE** (p. 513) otherwise.

9.253.2.67 get_typecode()

```
virtual const  DDS_TypeCode * DDSDomainParticipant::get_typecode (
             const char * type_name )  [pure virtual]
```

<<***extension***>> *(p. 432)* Retrieves the **DDS_TypeCode** (p. 1428) of a type registered with the **DDSDomain↩ Participant** (p. 1619) based on the registration name.

Every data type used in a **DDSDomainParticipant** (p. 1619) has a registered type name which is the name specified at the time the type is registered with the **DDSDomainParticipant** (p. 1619).

This operation retrieves the **DDS_TypeCode** (p. 1428) for the type given the registared type name. The `type_name` argument must correspond to a name used to register a type with the **DDSDomainParticipant** (p. 1619). Otherwise, this method will return NULL.

Type registration in a **DDSDomainParticipant** (p. 1619) is performed by a call to the **FooTypeSupport::register_type** (p. 1981) operation on a derived class of the **DDSTypeSupport** (p. 1897) interface.

Type registration might occur directly via an application call to the **FooTypeSupport::register_type** (p. 1981) operation, or indirectly by the RTI Connext infrastructure as a result of parsing an XML configuration file that refers to that type.

If a type is registered by the RTI Connext infrastructure as a result of parsing an XML configuration file, the **DDSType↩ Support** (p. 1897) can be created either from a type description found in the XML files, or else by calling the **FooTypeSupport::register_type** (p. 1981) operation on a **DDSTypeSupport** (p. 1897) that has been registered with **DDSDomainParticipantFactory** (p. 1694). If a **DDSTypeSupport** (p. 1897) is registered with the **DDSDomain↩ ParticipantFactory** (p. 1694) this mechanism takes precedence over the creation of a **DDSTypeSupport** (p. 1897) from a type decription in the XML file.

To register a **DDSTypeSupport** (p. 1897) with the **DDSDomainParticipantFactory** (p. 1694) use the operation **DDSDomainParticipantFactory::register_type_support** (p. 1720).

If the **DDSTypeSupport** (p. 1897) is created from a type description found in the XML files, the resulting type support will be a **DDSDynamicDataTypeSupport** (p. 1725). In this case the **DDSDataWriter** (p. 1589) and **DDSDataReader** (p. 1556) uses to write and read data of that type will be the **DDSDynamicDataWriter** (p. 1730) and **DDSDynamic↩ DataReader** (p. 1724), respectively.

Parameters

type_name	<<***in***>> *(p. 433)* Name of the type whose TypeCode is retrieved.

Returns

The TypeCode of a registered type with the **DDSDomainParticipant** (p. 1619) or NULL if no type was registered with the participant under that name.

9.253.2.68 get_participant_protocol_status()

```
virtual  DDS_ReturnCode_t DDSDomainParticipant::get_participant_protocol_status (
            struct  DDS_DomainParticipantProtocolStatus & status )  [pure virtual]
```

<<***extension***>> *(p. 432)* Get the domain participant protocol status for this participant.

This also resets the status so that it is no longer considered changed.

Parameters

status	<<***inout***>> *(p. 433)* **DDS_DomainParticipantProtocolStatus** (p. 970) to be filled in.

Exceptions

One	of the **Standard Return Codes** (p. 532) or **DDS_RETCODE_NOT_ENABLED** (p. 533).

9.253.2.69 set_property()

```
virtual  DDS_ReturnCode_t DDSDomainParticipant::set_property (
            const char * property_name,
            const char * value,
            bool propagate )  [pure virtual]
```

Set the value for a property that applies to a DomainParticipant.

Warning

This method is not implemented in all APIs and it's intended only for testing purposes. You should use **DDSDomainParticipant::set_qos** (p. 1677) instead.

Parameters

property_name	<<***in***>> *(p. 433)*. Name of the property that you want to set.
value	<<***in***>> *(p. 433)*. New value for the property.
propagate	<<***in***>> *(p. 433)*. Indicates if the property will be propagated or not.

Exceptions

One	of the **Standard Return Codes** (p. 532)

See also

DDSDataWriter::set_property (p. 1605)

DDSDataReader::set_property (p. 1576)

DDSDomainParticipant::set_qos (p. 1677)

9.253.2.70 set_qos()

```
virtual DDS_ReturnCode_t DDSDomainParticipant::set_qos (
            const DDS_DomainParticipantQos & qos ) [pure virtual]
```

Change the QoS of this DomainParticipant.

The **DDS_DomainParticipantQos::user_data** (p. 973) and **DDS_DomainParticipantQos::entity_factory** (p. 974) can be changed. The other policies are immutable.

Parameters

qos	<<***in***>> *(p. 433)* Set of policies to be applied to **DDSDomainParticipant** (p. 1619). Policies must be consistent. Immutable policies cannot be changed after **DDSDomainParticipant** (p. 1619) is enabled. The special value **DDS_PARTICIPANT_QOS_DEFAULT** (p. 50) can be used to indicate that the QoS of the **DDSDomainParticipant** (p. 1619) should be changed to match the current default **DDS_DomainParticipantQos** (p. 971) set in the **DDSDomainParticipantFactory** (p. 1694).

Exceptions

One	of the **Standard Return Codes** (p. 532), **DDS_RETCODE_IMMUTABLE_POLICY** (p. 533) if an immutable policy is changed, **DDS_RETCODE_INCONSISTENT_POLICY** (p. 533) if policies are inconsistent, or **DDS_RETCODE_NOT_ALLOWED_BY_SECURITY** (p. 533) if policies are not allowed by the Security Plugins

See also

DDS_DomainParticipantQos (p. 971) for rules on consistency among QoS policies

set_qos (abstract) (p. 1732)

9.253.2.71 set_qos_with_profile()

```
virtual DDS_ReturnCode_t DDSDomainParticipant::set_qos_with_profile (
            const char * library_name,
            const char * profile_name ) [pure virtual]
```

<<***extension***>> *(p. 432)* Change the QoS of this domain participant using the input XML QoS profile.

The **DDS_DomainParticipantQos::user_data** (p. 973) and **DDS_DomainParticipantQos::entity_factory** (p. 974) can be changed. The other policies are immutable.

Parameters

library_name	<<***in***>> *(p. 433)* Library name containing the XML QoS profile. If library_name is null RTI Connext will use the default library (see **DDSDomainParticipantFactory::set_default_library** (p. 1701)).
profile_name	<<***in***>> *(p. 433)* XML QoS Profile name. If profile_name is null RTI Connext will use the default profile (see **DDSDomainParticipantFactory::set_default_profile** (p. 1702)).

Exceptions

One	of the **Standard Return Codes** (p. 532), **DDS_RETCODE_IMMUTABLE_POLICY** (p. 533) if immutable policy is changed, or **DDS_RETCODE_INCONSISTENT_POLICY** (p. 533) if policies are inconsistent

See also

DDS_DomainParticipantQos (p. 971) for rules on consistency among QoS

9.253.2.72 get_qos()

```
virtual DDS_ReturnCode_t DDSDomainParticipant::get_qos (
            DDS_DomainParticipantQos & qos ) [pure virtual]
```

Get the participant QoS.

This method may potentially allocate memory depending on the sequences contained in some QoS policies.

Parameters

qos	<<***inout***>> *(p. 433)* QoS to be filled up.

Exceptions

One	of the **Standard Return Codes** (p. 532)

See also

get_qos (abstract) (p. 1733)

9.253.2.73 add_peer()

```
virtual DDS_ReturnCode_t DDSDomainParticipant::add_peer (
            const char * peer_desc_string ) [pure virtual]
```

<<***extension***>> *(p. 432)* Attempt to contact one or more additional peer participants.

Add the given peer description to the list of peers with which this **DDSDomainParticipant** (p. 1619) will try to communicate.

This method may be called at any time after this **DDSDomainParticipant** (p. 1619) has been created (before or after it has been enabled).

If this method is called after **DDSEntity::enable** (p. 1734), an attempt will be made to contact the new peer(s) immediately.

If this method is called *before* the DomainParticipant is enabled, the peer description will simply be added to the list that was populated by **DDS_DiscoveryQosPolicy::initial_peers** (p. 962); the first attempted contact will take place after this **DDSDomainParticipant** (p. 1619) is enabled.

Any peers added with the add_peer operation are equivalent to entries in the initial_peers list. Therefore, the participant will only discover remote participants using the newly added peer descriptor if a remote participant exists at that location and is configured to accept new peers (i.e., **DDS_DiscoveryQosPolicy::accept_unknown_peers** (p. 963) is set to **DDS_BOOLEAN_TRUE** (p. 513)). If multiple remote participants exist at that location (for example, if add_peer is called with a multicast address), the participant may discover multiple remote participants.

To be informed of the exact remote participants that are discovered, regardless of which peers this **DDSDomain↩ Participant** (p. 1619) *attempts* to discover, use the built-in participant topic: **DDS_PARTICIPANT_TOPIC_NAME** (p. 491).

To remove specific peer locators, you may use **DDSDomainParticipant::remove_peer** (p. 1679). If a peer is removed, the add_peer operation will add it back to the list of peers.

To stop communicating with a peer **DDSDomainParticipant** (p. 1619) that has been discovered, use **DDSDomain↩ Participant::ignore_participant** (p. 1661).

Adding a peer description with this method has no effect on the **DDS_DiscoveryQosPolicy::initial_peers** (p. 962) that may be subsequently retrieved with **DDSDomainParticipant::get_qos()** (p. 1678) (because **DDS_DiscoveryQos↩ Policy** (p. 960) is immutable).

Parameters

peer_desc_string	<<***in***>> *(p. 433)* New peer descriptor to be added. The format is specified in **Peer Descriptor Format** (p. 663).

Exceptions

One	of the **Standard Return Codes** (p. 532)

See also

Peer Descriptor Format (p. 663)

DDS_DiscoveryQosPolicy::initial_peers (p. 962)

DDS_PARTICIPANT_TOPIC_NAME (p. 491)

DDSDomainParticipant::get_builtin_subscriber (p. 1661)

9.253.2.74 remove_peer()

```
virtual DDS_ReturnCode_t DDSDomainParticipant::remove_peer (
            const char * peer_desc_string ) [pure virtual]
```

<<***extension***>> *(p. 432)* Remove one or more peer participants from the list of peers with which this **DDSDomainParticipant** (p. 1619) will try to communicate.

This method may be called any time after this **DDSDomainParticipant** (p. 1619) has been enabled

Calling this method has the following effects:

- If a **DDSDomainParticipant** (p. 1619) was already discovered, it will be locally removed along with all its entities.
- Any further requests coming from a **DDSDomainParticipant** (p. 1619) located on any of the removed peers will be ignored.
- All the locators contained in the peer description will be removed from the peer list. The local **DDSDomainParticipant** (p. 1619) will stop sending announcement to those locators.

If remote participants located on a peer that was previously removed are discovered, they will be ignored untill the related peer is added back by using **DDSDomainParticipant::add_peer** (p. 1678).

Removing a peer description with this method has no effect on the **DDS_DiscoveryQosPolicy::initial_peers** (p. 962) that may be subsequently retrieved with **DDSDomainParticipant::get_qos()** (p. 1678) (because **DDS_DiscoveryQosPolicy** (p. 960) is immutable).

Parameters

peer_desc_string	<<***in***>> *(p. 433)* Peer descriptor to be removed. The format is specified in **Peer Descriptor Format** (p. 663).

Exceptions

One	of the **Standard Return Codes** (p. 532)

See also

Peer Descriptor Format (p. 663)

DDS_DiscoveryQosPolicy::initial_peers (p. 962)

DDSDomainParticipant::add_peer (p. 1678)

9.253.2.75 get_dns_tracker_polling_period()

```
virtual DDS_ReturnCode_t DDSDomainParticipant::get_dns_tracker_polling_period (
            DDS_Duration_t & polling_period ) [pure virtual]
```

*<<**extension**>> (p. 432)* Retrieves the frequency used by the DNS tracker thread to query the DNS service.

The DNS tracker queries the DNS for hostnames specified in the initial peers of a DomainParticipant. The frequency of these queries is defined by **DDS_DiscoveryConfigQosPolicy::dns_tracker_polling_period** (p. 956). If the value returned is **DDS_DURATION_INFINITE** (p. 522), the DNS tracker is disabled.

Parameters

polling_period	*<<**out**>> (p. 433)* Duration that the API populates with the period of the DNS tracker.

Exceptions

One	of the **Standard Return Codes** (p. 532)

See also

DDS_DiscoveryConfigQosPolicy::dns_tracker_polling_period (p. 956)

9.253.2.76 set_dns_tracker_polling_period()

```
virtual DDS_ReturnCode_t DDSDomainParticipant::set_dns_tracker_polling_period (
            const DDS_Duration_t & polling_period ) [pure virtual]
```

*<<**extension**>> (p. 432)* Configures the frequency in which the DNS tracker queries the DNS service.

This API allows you to change the frequency of the polling period for the DNS tracker. The range of accepted values, in seconds, goes from 1 second to 1 year. **DDS_DURATION_INFINITE** (p. 522) is also accepted as a valid value. If the duration is set to **DDS_DURATION_INFINITE** (p. 522), the DNS tracker is disabled.

Modifying the DNS tracker polling period through this has no effect on the **DDS_DiscoveryConfigQosPolicy::dns_↩ tracker_polling_period** (p. 956) when it is retrieved with **DDSDomainParticipant::get_qos()** (p. 1678).

Parameters

polling_period	*<<**in**>> (p. 433)* Duration that is set as the polling period for the DNS tracker.

Exceptions

One	of the **Standard Return Codes** (p. 532)

See also

DDS_DiscoveryConfigQosPolicy::dns_tracker_polling_period (p. 956)

DDSDomainParticipant::add_peer (p. 1678)

9.253.2.77 set_listener()

```
virtual DDS_ReturnCode_t DDSDomainParticipant::set_listener (
            DDSDomainParticipantListener * l,
            DDS_StatusMask mask = DDS_STATUS_MASK_ALL ) [pure virtual]
```

Sets the participant listener.

Parameters

l	<<***in***>> *(*p. *433)* Listener to be installed on the entity.
mask	<<***in***>> *(*p. *433)* Changes of communication status to be invoked on the listener. See **DDS_StatusMask** (p. 537).

MT Safety:

Unsafe. This method is not synchronized with the listener callbacks, so it is possible to set a new listener on a participant when the old listener is in a callback.

Care should therefore be taken not to delete any listener that has been set on an enabled participant unless some application-specific means are available of ensuring that the old listener cannot still be in use.

Exceptions

One	of the **Standard Return Codes** (p. 532)

See also

set_listener (abstract) (p. 1733)

9.253.2.78 get_listener()

```
virtual DDSDomainParticipantListener * DDSDomainParticipant::get_listener ( ) [pure virtual]
```

Get the participant listener.

Returns

Existing listener attached to the **DDSDomainParticipant** (p. 1619).

See also

get_listener (abstract) (p. 1734)

9.253.2.79 get_implicit_publisher()

```
virtual DDSPublisher * DDSDomainParticipant::get_implicit_publisher ( ) [pure virtual]
```

<<***extension***>> *(p. 432)* Returns the implicit **DDSPublisher** (p. 1818). If an implicit Publisher does not already exist, this creates one.

There can only be one implicit Publisher per DomainParticipant.

The implicit Publisher is created with **DDS_PUBLISHER_QOS_DEFAULT** (p. 59) and no Listener.

This implicit Publisher will be deleted automatically when the following methods are called: **DDSDomainParticipant::delete_contained_entities** (p. 1669), or **DDSDomainParticipant::delete_publisher** (p. 1646) with the implicit publisher as a parameter. Additionally, when a DomainParticipant is deleted, if there are no attached DataWriters that belong to the implicit Publisher, the implicit Publisher will be implicitly deleted.

Returns

The implicit publisher

See also

DDS_PUBLISHER_QOS_DEFAULT (p. 59)

DDSDomainParticipant::create_publisher (p. 1644)

9.253.2.80 get_implicit_subscriber()

```
virtual DDSSubscriber * DDSDomainParticipant::get_implicit_subscriber ( ) [pure virtual]
```

<<***extension***>> *(p. 432)* Returns the implicit **DDSSubscriber** (p. 1860). If an implicit Subscriber does not already exist, this creates one.

There can only be one implicit Subscriber per DomainParticipant.

The implicit Subscriber is created with **DDS_SUBSCRIBER_QOS_DEFAULT** (p. 59) and no Listener.

This implicit Subscriber will be deleted automatically when the following methods are called: **DDSDomainParticipant::delete_contained_entities** (p. 1669), or **DDSDomainParticipant::delete_subscriber** (p. 1648) with the subscriber as a parameter. Additionally, when a DomainParticipant is deleted, if there are no attached DataReaders that belong to the implicit Subscriber, the implicit Subscriber will be implicitly deleted.

MT Safety:

UNSAFE. it is not safe to create the implicit subscriber while another thread may be simultaneously calling **DDSDomainParticipant::set_default_subscriber_qos** (p. 1642).

Returns

The implicit subscriber

See also

DDS_SUBSCRIBER_QOS_DEFAULT (p. 59)

DDSDomainParticipant::create_subscriber (p. 1647)

9.253.2.81 create_datawriter()

```
virtual DDSDataWriter * DDSDomainParticipant::create_datawriter (
            DDSTopic * topic,
          const DDS_DataWriterQos & qos,
            DDSDataWriterListener * listener,
            DDS_StatusMask mask ) [pure virtual]
```

<<***extension***>> *(*p. *432)* Creates a **DDSDataWriter** (p. 1589) that will be attached and belong to the implicit **DDSPublisher** (p. 1818).

Precondition

The given **DDSTopic** (p. 1885) must have been created from the same DomainParticipant as the implicit Publisher. If it was created from a different DomainParticipant, this method will fail.

The **DDSDataWriter** (p. 1589) created using this method will be associated with the implicit Publisher. This Publisher is automatically created (if it does not exist) using **DDS_PUBLISHER_QOS_DEFAULT** (p. 59) when the following methods are called: **DDSDomainParticipant::create_datawriter** (p. 1683), **DDSDomainParticipant::create_datawriter_with_profile** (p. 1685), or **DDSDomainParticipant::get_implicit_publisher** (p. 1682).

MT Safety:

UNSAFE. If **DDS_DATAWRITER_QOS_DEFAULT** (p. 114) is used for the `qos` parameter, it is not safe to create the DataWriter while another thread may be simultaneously calling **DDSDomainParticipant::set_default_datawriter_qos** (p. 1627).

Parameters

topic	<<***in***>> *(*p. *433)* The **DDSTopic** (p. 1885) that the **DDSDataWriter** (p. 1589) will be associated with.
qos	<<***in***>> *(*p. *433)* QoS to be used for creating the new **DDSDataWriter** (p. 1589). The special value **DDS_DATAWRITER_QOS_DEFAULT** (p. 114) can be used to indicate that the **DDSDataWriter** (p. 1589) should be created with the default **DDS_DataWriterQos** (p. 909) set in the implicit **DDSPublisher** (p. 1818). The special value **DDS_DATAWRITER_QOS_USE_TOPIC_QOS** (p. 114) can be used to indicate that the **DDSDataWriter** (p. 1589) should be created with the combination of the default **DDS_DataWriterQos** (p. 909) set on the **DDSPublisher** (p. 1818) and the **DDS_TopicQos** (p. 1397) of the **DDSTopic** (p. 1885).
listener	<<***in***>> *(*p. *433)* The listener of the **DDSDataWriter** (p. 1589).
mask	<<***in***>> *(*p. *433).* Changes of communication status to be invoked on the listener. See **DDS_StatusMask** (p. 537).

Returns

A **DDSDataWriter** (p. 1589) of a derived class specific to the data type associated with the **DDSTopic** (p. 1885) or NULL if an error occurred.

See also

FooDataWriter (p. 1944)

Specifying QoS on entities (p. 555) for information on setting QoS before entity creation

DDS_DataWriterQos (p. 909) for rules on consistency among QoS

DDS_DATAWRITER_QOS_DEFAULT (p. 114)

DDS_DATAWRITER_QOS_USE_TOPIC_QOS (p. 114)

DDSDomainParticipant::create_datawriter_with_profile (p. 1685)

DDSDomainParticipant::get_default_datawriter_qos (p. 1626)

DDSDomainParticipant::get_implicit_publisher (p. 1682)

DDSTopic::set_qos (p. 1887)

DDSDataWriter::set_listener (p. 1607)

9.253.2.82 create_datawriter_with_profile()

```
virtual DDSDataWriter * DDSDomainParticipant::create_datawriter_with_profile (
            DDSTopic * topic,
           const char * library_name,
           const char * profile_name,
            DDSDataWriterListener * listener,
            DDS_StatusMask mask )  [pure virtual]
```

<<***extension***>> *(p. 432)* Creates a **DDSDataWriter** (p. 1589) using a XML QoS profile that will be attached and belong to the implicit **DDSPublisher** (p. 1818).

Precondition

The given **DDSTopic** (p. 1885) must have been created from the same DomainParticipant as the implicit Publisher. If it was created from a different DomainParticipant, this method will return NULL.

The **DDSDataWriter** (p. 1589) created using this method will be associated with the implicit Publisher. This Publisher is automatically created (if it does not exist) using **DDS_PUBLISHER_QOS_DEFAULT** (p. 59) when the following methods are called: **DDSDomainParticipant::create_datawriter** (p. 1683), **DDSDomainParticipant::create_datawriter_with_profile** (p. 1685), or **DDSDomainParticipant::get_implicit_publisher** (p. 1682)

Parameters

topic	<<***in***>> *(p. 433)* The **DDSTopic** (p. 1885) that the **DDSDataWriter** (p. 1589) will be associated with.
library_name	<<***in***>> *(p. 433)* Library name containing the XML QoS profile. If library_name is null RTI Connext will use the default library (see **DDSDomainParticipant::set_default_library** (p. 1635)).
profile_name	<<***in***>> *(p. 433)* XML QoS Profile name. If profile_name is null RTI Connext will use the default profile (see **DDSDomainParticipant::set_default_profile** (p. 1636)).
listener	<<***in***>> *(p. 433)* The listener of the **DDSDataWriter** (p. 1589).
mask	<<***in***>> *(p. 433).* Changes of communication status to be invoked on the listener. See **DDS_StatusMask** (p. 537).

Returns

A **DDSDataWriter** (p. 1589) of a derived class specific to the data type associated with the **DDSTopic** (p. 1885) or NULL if an error occurred.

See also

FooDataWriter (p. 1944)

Specifying QoS on entities (p. 555) for information on setting QoS before entity creation

DDS_DataWriterQos (p. 909) for rules on consistency among QoS

DDSDomainParticipant::create_datawriter (p. 1683)

DDSDomainParticipant::get_default_datawriter_qos (p. 1626)

DDSDomainParticipant::get_implicit_publisher (p. 1682)

DDSTopic::set_qos (p. 1887)

DDSDataWriter::set_listener (p. 1607)

9.253.2.83 delete_datawriter()

```
virtual  DDS_ReturnCode_t DDSDomainParticipant::delete_datawriter (
              DDSDataWriter * a_datawriter )  [pure virtual]
```

<<***extension***>> *(p. 432)* Deletes a **DDSDataWriter** (p. 1589) that belongs to the implicit **DDSPublisher** (p. 1818).

The deletion of the **DDSDataWriter** (p. 1589) will automatically unregister all instances.

Precondition

If the **DDSDataWriter** (p. 1589) does not belong to the implicit **DDSPublisher** (p. 1818), the operation will fail with **DDS_RETCODE_PRECONDITION_NOT_MET** (p. 532).

Postcondition

Listener installed on the **DDSDataWriter** (p. 1589) will not be called after this method completes successfully.

MT Safety:

UNSAFE. It is not safe to delete an entity while another thread may be simultaneously calling an API that uses the entity.

Parameters

a_datawriter	<<***in***>> *(p. 433)* The **DDSDataWriter** (p. 1589) to be deleted.

Exceptions

One	of the **Standard Return Codes** (p. 532) or **DDS_RETCODE_PRECONDITION_NOT_MET** (p. 532).

See also

DDSDomainParticipant::get_implicit_publisher (p. 1682)

9.253.2.84 create_datareader()

```
virtual DDSDataReader * DDSDomainParticipant::create_datareader (
            DDSTopicDescription * topic,
          const DDS_DataReaderQos & qos,
            DDSDataReaderListener * listener,
            DDS_StatusMask mask ) [pure virtual]
```

<<***extension***>> *(p. 432)* Creates a **DDSDataReader** (p. 1556) that will be attached and belong to the implicit **DDSSubscriber** (p. 1860).

Precondition

The given **DDSTopicDescription** (p. 1892) must have been created from the same DomainParticipant as the implicit Subscriber. If it was created from a different DomainParticipant, this method will return NULL.

The **DDSDataReader** (p. 1556) created using this method will be associated with the implicit Subscriber. This Subscriber is automatically created (if it does not exist) using **DDS_SUBSCRIBER_QOS_DEFAULT** (p. 59) when the following methods are called: **DDSDomainParticipant::create_datareader** (p. 1687), **DDSDomainParticipant::create_datareader_with_profile** (p. 1688), or **DDSDomainParticipant::get_implicit_subscriber** (p. 1683).

MT Safety:

UNSAFE. If **DDS_DATAREADER_QOS_DEFAULT** (p. 136) is used for the `qos` parameter, it is not safe to create the datareader while another thread may be simultaneously calling **DDSDomainParticipant::set_default_datareader_qos** (p. 1629).

Parameters

topic	<<***in***>> *(p. 433)* The **DDSTopicDescription** (p. 1892) that the **DDSDataReader** (p. 1556) will be associated with.
qos	<<***in***>> *(p. 433)* The qos of the **DDSDataReader** (p. 1556). The special value **DDS_DATAREADER_QOS_DEFAULT** (p. 136) can be used to indicate that the **DDSDataReader** (p. 1556) should be created with the default **DDS_DataReaderQos** (p. 858) set in the implicit **DDSSubscriber** (p. 1860). If **DDSTopicDescription** (p. 1892) is of type **DDSTopic** (p. 1885) or **DDSContentFilteredTopic** (p. 1551), the special value **DDS_DATAREADER_QOS_USE_TOPIC_QOS** (p. 137) can be used to indicate that the **DDSDataReader** (p. 1556) should be created with the combination of the default **DDS_DataReaderQos** (p. 858) set on the implicit **DDSSubscriber** (p. 1860) and the **DDS_TopicQos** (p. 1397) (in the case of a **DDSContentFilteredTopic** (p. 1551), the **DDS_TopicQos** (p. 1397) of the related **DDSTopic** (p. 1885)). if **DDS_DATAREADER_QOS_USE_TOPIC_QOS** (p. 137) is used, `topic` cannot be a **DDSMultiTopic** (p. 1797).
listener	<<***in***>> *(p. 433)* The listener of the **DDSDataReader** (p. 1556).
mask	<<***in***>> *(p. 433)* Changes of communication status to be invoked on the listener. See

Returns

A **DDSDataReader** (p. 1556) of a derived class specific to the data-type associated with the **DDSTopic** (p. 1885) or NULL if an error occurred.

See also

FooDataReader (p. 1917)

Specifying QoS on entities (p. 555) for information on setting QoS before entity creation

DDS_DataReaderQos (p. 858) for rules on consistency among QoS

DDSDomainParticipant::create_datareader_with_profile (p. 1688)

DDSDomainParticipant::get_default_datareader_qos (p. 1628)

DDSDomainParticipant::get_implicit_subscriber (p. 1683)

DDSTopic::set_qos (p. 1887)

DDSDataReader::set_listener (p. 1577)

9.253.2.85 create_datareader_with_profile()

```
virtual DDSDataReader * DDSDomainParticipant::create_datareader_with_profile (
            DDSTopicDescription * topic,
          const char * library_name,
          const char * profile_name,
            DDSDataReaderListener * listener,
            DDS_StatusMask mask ) [pure virtual]
```

<<***extension***>> *(p. 432)* Creates a **DDSDataReader** (p. 1556) using a XML QoS profile that will be attached and belong to the implicit **DDSSubscriber** (p. 1860).

Precondition

The given **DDSTopicDescription** (p. 1892) must have been created from the same DomainParticipant as the implicit subscriber. If it was created from a different DomainParticipant, this method will return NULL.

The **DDSDataReader** (p. 1556) created using this method will be associated with the implicit Subscriber. This Subscriber is automatically created (if it does not exist) using **DDS_SUBSCRIBER_QOS_DEFAULT** (p. 59) when the following methods are called: **DDSDomainParticipant::create_datareader** (p. 1687), **DDSDomainParticipant::create_datareader_with_profile** (p. 1688), or **DDSDomainParticipant::get_implicit_subscriber** (p. 1683)

Parameters

topic	<<***in***>> *(p. 433)* The **DDSTopicDescription** (p. 1892) that the **DDSDataReader** (p. 1556) will be associated with.
library_name	<<***in***>> *(p. 433)* Library name containing the XML QoS profile. If library_name is null RTI Connext will use the default library (see **DDSDomainParticipant::set_default_library** (p. 1635)).
profile_name	<<***in***>> *(p. 433)* XML QoS Profile name. If profile_name is null RTI Connext will use the default profile (see **DDSDomainParticipant::set_default_profile** (p. 1636)).
listener	<<***in***>> *(p. 433)* The listener of the **DDSDataReader** (p. 1556).
mask	<<***in***>> *(p. 433)*. Changes of communication status to be invoked on the listener. See **DDS_StatusMask** (p. 537).

Returns

A **DDSDataReader** (p. 1556) of a derived class specific to the data-type associated with the **DDSTopic** (p. 1885) or NULL if an error occurred.

See also

FooDataReader (p. 1917)

Specifying QoS on entities (p. 555) for information on setting QoS before entity creation

DDS_DataReaderQos (p. 858) for rules on consistency among QoS

DDSDomainParticipant::create_datareader (p. 1687)

DDSDomainParticipant::get_default_datareader_qos (p. 1628)

DDSDomainParticipant::get_implicit_subscriber (p. 1683)

DDSTopic::set_qos (p. 1887)

DDSDataReader::set_listener (p. 1577)

9.253.2.86 delete_datareader()

```
virtual  DDS_ReturnCode_t DDSDomainParticipant::delete_datareader (
              DDSDataReader * a_datareader )  [pure virtual]
```

<<***extension***>> *(p. 432)* Deletes a **DDSDataReader** (p. 1556) that belongs to the implicit **DDSSubscriber** (p. 1860).

Precondition

If the **DDSDataReader** (p. 1556) does not belong to the implicit **DDSSubscriber** (p. 1860), or if ther are any existing **DDSReadCondition** (p. 1842) or **DDSQueryCondition** (p. 1841) objects that are attached to the **DDSDataReader** (p. 1556), or if there are outstanding loans on samples (as a result of a call to read(), take(), or one of the variants thereof), the operation fails with the error **DDS_RETCODE_PRECONDITION_NOT_MET** (p. 532).

Postcondition

Listener installed on the **DDSDataReader** (p. 1556) will not be called after this method completes successfully.

MT Safety:

UNSAFE. It is not safe to delete an entity while another thread may be simultaneously calling an API that uses the entity.

Parameters

a_datareader	<<***in***>> *(p. 433)* The **DDSDataReader** (p. 1556) to be deleted.

Exceptions

One	of the **Standard Return Codes** (p. 532) or **DDS_RETCODE_PRECONDITION_NOT_MET** (p. 532).

See also

DDSDomainParticipant::get_implicit_subscriber (p. 1683)

9.253.2.87 lookup_publisher_by_name()

```
virtual DDSPublisher * DDSDomainParticipant::lookup_publisher_by_name (
            const char * publisher_name ) [pure virtual]
```

<<***extension***>> *(p. 432)* Looks up a **DDSPublisher** (p. 1818) by its entity name within this **DDSDomainParticipant** (p. 1619).

Every **DDSPublisher** (p. 1818) in the system has an entity name which is configured and stored in the EntityName policy, **ENTITY_NAME** (p. 601).

This operation retrieves a **DDSPublisher** (p. 1818) within the **DDSDomainParticipant** (p. 1619) given the entity's name. If there are several **DDSPublisher** (p. 1818) with the same name within the **DDSDomainParticipant** (p. 1619), this function returns the first matching occurrence.

Parameters

publisher_name	<<***in***>> *(p. 433)* Entity name of the **DDSPublisher** (p. 1818).

Returns

The first **DDSPublisher** (p. 1818) found with the specified name or NULL if it is not found.

See also

DDSDomainParticipant::lookup_datawriter_by_name (p. 1691)

9.253.2.88 lookup_subscriber_by_name()

```
virtual DDSSubscriber * DDSDomainParticipant::lookup_subscriber_by_name (
            const char * subscriber_name ) [pure virtual]
```

<<***extension***>> *(p. 432)* Retrieves a **DDSSubscriber** (p. 1860) by its entity name within this **DDSDomainParticipant** (p. 1619).

Every **DDSSubscriber** (p. 1860) in the system has an entity name which is configured and stored in the EntityName policy, **ENTITY_NAME** (p. 601).

This operation retrieves a **DDSSubscriber** (p. 1860) within the **DDSDomainParticipant** (p. 1619) given the entity's name. If there are several **DDSSubscriber** (p. 1860) with the same name within the **DDSDomainParticipant** (p. 1619), this function returns the first matching occurrence.

Parameters

subscriber_name	<<***in***>> *(p. 433)* Entity name of the **DDSSubscriber** (p. 1860).

Returns

The first **DDSSubscriber** (p. 1860) found with the specified name or NULL if it is not found.

See also

DDSDomainParticipant::lookup_datareader_by_name (p. 1692)

9.253.2.89 lookup_datawriter_by_name()

```
virtual DDSDataWriter * DDSDomainParticipant::lookup_datawriter_by_name (
            const char * datawriter_full_name ) [pure virtual]
```

<<***extension***>> *(p. 432)* Looks up a **DDSDataWriter** (p. 1589) by its entity name within this **DDSDomainParticipant** (p. 1619).

Every **DDSDataWriter** (p. 1589) in the system has an entity name which is configured and stored in the EntityName policy, **ENTITY_NAME** (p. 601).

Every **DDSPublisher** (p. 1818) in the system has an entity name which is also configured and stored in the EntityName policy.

This operation retrieves a **DDSDataWriter** (p. 1589) within a **DDSPublisher** (p. 1818) given the specified name which encodes both to the **DDSDataWriter** (p. 1589) and the **DDSPublisher** (p. 1818) name.

If there are several **DDSDataWriter** (p. 1589) with the same name within the corresponding **DDSPublisher** (p. 1818) this function returns the first matching occurrence.

The specified name might be given as a fully-qualified entity name or as a plain name.

The fully qualified entity name is a concatenation of the **DDSPublisher** (p. 1818) to which the **DDSDataWriter** (p. 1589) belongs and the entity name of of the **DDSDataWriter** (p. 1589) itself, separated by a double colon "::". For example: MyPublisherName::MyDataWriterName

The plain name contains the **DDSDataWriter** (p. 1589) name only. In this situation it is implied that the **DDSDataWriter** (p. 1589) belongs to the implicit **DDSPublisher** (p. 1818) so the use of a plain name is equivalent to specifying a fully qualified name with the **DDSPublisher** (p. 1818) name part being "implicit". For example: the plain name "MyData↩ WriterName" is equivalent to specifiying the fully qualified name "implicit::MyDataWriterName"

The **DDSDataWriter** (p. 1589) is only looked up within the **DDSPublisher** (p. 1818) specified in the fully qualified name, or within the implicit **DDSPublisher** (p. 1818) if the name was not fully qualified.

Parameters

datawriter_full_name	<<***in***>> *(*p. *433)* Entity name or fully-qualified entity name of the **DDSDataWriter** (p. 1589).

Returns

The first **DDSDataWriter** (p. 1589) found with the specified name or NULL if it is not found.

See also

DDSPublisher::lookup_datawriter_by_name (p. 1839)

DDSDomainParticipant::lookup_publisher_by_name (p. 1690)

9.253.2.90 lookup_datareader_by_name()

```
virtual DDSDataReader * DDSDomainParticipant::lookup_datareader_by_name (
            const char * datareader_full_name ) [pure virtual]
```

<<***extension***>> *(*p. *432)* Retrieves up a **DDSDataReader** (p. 1556) by its entity name in this **DDSDomainParticipant** (p. 1619).

Every **DDSDataReader** (p. 1556) in the system has an entity name which is configured and stored in the EntityName policy, **ENTITY_NAME** (p. 601).

Every **DDSSubscriber** (p. 1860) in the system has an entity name which is also configured and stored in the EntityName policy, **ENTITY_NAME** (p. 601).

This operation retrieves a **DDSDataReader** (p. 1556) within a **DDSSubscriber** (p. 1860) given the specified name which encodes both to the **DDSDataReader** (p. 1556) and the **DDSSubscriber** (p. 1860) name.

If there are several **DDSDataReader** (p. 1556) with the same name within the corresponding **DDSSubscriber** (p. 1860) this function returns the first matching occurrence.

The specified name might be given as a fully-qualified entity name or as a plain name.

The fully qualified entity name is a concatenation of the **DDSSubscriber** (p. 1860) to which the **DDSDataReader** (p. 1556) belongs and the entity name of of the **DDSDataReader** (p. 1556) itself, separated by a double colon "::". For example: MySubscriberName::MyDataReaderName

The plain name contains the **DDSDataReader** (p. 1556) name only. In this situation it is implied that the **DDSDataReader** (p. 1556) belongs to the implicit **DDSSubscriber** (p. 1860) so the use of a plain name is equivalent to specifying a fully qualified name with the **DDSSubscriber** (p. 1860) name part being "implicit". For example: the plain name "MyDataReaderName" is equivalent to specifiying the fully qualified name "implicit::MyDataReaderName"

The **DDSDataReader** (p. 1556) is only looked up within the **DDSSubscriber** (p. 1860) specified in the fully qualified name, or within the implicit **DDSSubscriber** (p. 1860) if the name was not fully qualified.

Parameters

datareader_full_name	<<***in***>> *(p. 433)* Full entity name of the **DDSDataReader** (p. 1556).

Returns

The first **DDSDataReader** (p. 1556) found with the specified name or NULL if it is not found.

See also

DDSSubscriber::lookup_datareader_by_name (p. 1880)

DDSDomainParticipant::lookup_subscriber_by_name (p. 1690)

9.253.2.91 take_discovery_snapshot() [1/2]

```
virtual DDS_ReturnCode_t DDSDomainParticipant::take_discovery_snapshot ( ) [pure virtual]
```

Take a snapshot of the remote participants discovered by a local one.

The snapshot will be printed through the **NDDSConfigLogger** (p. 2093). A possible output may be the following:

```
Remote participants that match the local participant domain=0

name="participantTestName" role="participantTestRole" id="1"

guid_prefix="0x0101D8D1,0x20B83C0D,0x4495246E"

---------------------------------------------------------------------------

1. 0x0101542A,0x2C59B595,0xA1693BDF name="participantTestName"

role="participantTestRole"

unicastLocators="udpv4://192.168.1.170:7411"

---------------------------------------------------------------------------
```

Exceptions

One	of the **Standard Return Codes** (p. 532).

9.253.2.92 take_discovery_snapshot() [2/2]

```
virtual DDS_ReturnCode_t DDSDomainParticipant::take_discovery_snapshot (
            const char * file_name ) [pure virtual]
```

Take a snapshot of the remote participants discovered by a local one.

The snapshot will be printed in the file specified by file_name. A possible output may be the following:

```
Remote participants that match the local participant domain=0

name="participantTestName" role="participantTestRole" id="1"

guid_prefix="0x0101D8D1,0x20B83C0D,0x4495246E"

---------------------------------------------------------------------------

1. 0x0101542A,0x2C59B595,0xA1693BDF name="participantTestName"

role="participantTestRole"

unicastLocators="udpv4://192.168.1.170:7411"

---------------------------------------------------------------------------
```

Parameters

file_name	<<***in***>> *(*p. *433)* Name of the file where snapshot should be printed.

Exceptions

One	of the **Standard Return Codes** (p. 532).

9.254 DDSDomainParticipantFactory Class Reference

<<***singleton***>> *(*p. *433)* <<***interface***>> *(*p. *433)* Allows creation and destruction of **DDSDomainParticipant** (p. 1619) objects.

Public Member Functions

- virtual **DDS_ReturnCode_t set_default_participant_qos** (const **DDS_DomainParticipantQos** &qos)=0

  *Sets the default **DDS_DomainParticipantQos** (*p. *971) values for this domain participant factory.*
- virtual **DDS_ReturnCode_t set_default_participant_qos_with_profile** (const char ∗library_name, const char ∗profile_name)=0

  <<***extension***>> *(*p. *432) Sets the default **DDS_DomainParticipantQos** (*p. *971) values for this domain participant factory based on the input XML QoS profile.*
- virtual **DDS_ReturnCode_t get_default_participant_qos** (**DDS_DomainParticipantQos** &qos)=0

  *Initializes the **DDS_DomainParticipantQos** (*p. *971) instance with default values.*
- virtual **DDS_ReturnCode_t set_default_library** (const char ∗library_name)=0

  <<***extension***>> *(*p. *432) Sets the default XML library for a **DDSDomainParticipantFactory** (*p. *1694).*
- virtual const char ∗ **get_default_library** ()=0

  <<***extension***>> *(*p. *432) Gets the default XML library associated with a **DDSDomainParticipantFactory** (*p. *1694).*
- virtual **DDS_ReturnCode_t set_default_profile** (const char ∗library_name, const char ∗profile_name)=0

  <<***extension***>> *(*p. *432) Sets the default XML profile for a **DDSDomainParticipantFactory** (*p. *1694).*
- virtual const char ∗ **get_default_profile** ()=0

  <<***extension***>> *(*p. *432) Gets the default XML profile associated with a **DDSDomainParticipantFactory** (*p. *1694).*
- virtual const char ∗ **get_default_profile_library** ()=0

  <<***extension***>> *(*p. *432) Gets the library where the default XML profile is contained for a **DDSDomainParticipant↩ Factory** (*p. *1694).*

- virtual **DDS_ReturnCode_t get_participant_factory_qos_from_profile** (**DDS_DomainParticipantFactory↩ Qos** &qos, const char ∗library_name, const char ∗profile_name)=0

  *<<**extension**>> (p. 432) Gets the **DDS_DomainParticipantFactoryQos** (p. 967) values associated with the input XML QoS profile.*
- virtual **DDS_ReturnCode_t get_participant_qos_from_profile** (**DDS_DomainParticipantQos** &qos, const char ∗library_name, const char ∗profile_name)=0

  *<<**extension**>> (p. 432) Gets the **DDS_DomainParticipantQos** (p. 971) values associated with the input XML QoS profile.*
- virtual **DDS_ReturnCode_t get_publisher_qos_from_profile** (**DDS_PublisherQos** &qos, const char ∗library_name, const char ∗profile_name)=0

  *<<**extension**>> (p. 432) Gets the **DDS_PublisherQos** (p. 1276) values associated with the input XML QoS profile.*
- virtual **DDS_ReturnCode_t get_subscriber_qos_from_profile** (**DDS_SubscriberQos** &qos, const char ∗library_name, const char ∗profile_name)=0

  *<<**extension**>> (p. 432) Gets the **DDS_SubscriberQos** (p. 1367) values associated with the input XML QoS profile.*
- virtual **DDS_ReturnCode_t get_datawriter_qos_from_profile** (**DDS_DataWriterQos** &qos, const char ∗library_name, const char ∗profile_name)=0

  *<<**extension**>> (p. 432) Gets the **DDS_DataWriterQos** (p. 909) values associated with the input XML QoS profile.*
- virtual **DDS_ReturnCode_t get_datawriter_qos_from_profile_w_topic_name** (**DDS_DataWriterQos** &qos, const char ∗library_name, const char ∗profile_name, const char ∗topic_name)=0

  *<<**extension**>> (p. 432) Gets the **DDS_DataWriterQos** (p. 909) values associated with the input XML QoS profile while applying topic filters to the input topic name.*
- virtual **DDS_ReturnCode_t get_datareader_qos_from_profile** (**DDS_DataReaderQos** &qos, const char ∗library_name, const char ∗profile_name)=0

  *<<**extension**>> (p. 432) Gets the **DDS_DataReaderQos** (p. 858) values associated with the input XML QoS profile.*
- virtual **DDS_ReturnCode_t get_datareader_qos_from_profile_w_topic_name** (**DDS_DataReaderQos** &qos, const char ∗library_name, const char ∗profile_name, const char ∗topic_name)=0

  *<<**extension**>> (p. 432) Gets the **DDS_DataReaderQos** (p. 858) values associated with the input XML QoS profile while applying topic filters to the input topic name.*
- virtual **DDS_ReturnCode_t get_topic_qos_from_profile** (**DDS_TopicQos** &qos, const char ∗library_name, const char ∗profile_name)=0

  *<<**extension**>> (p. 432) Gets the **DDS_TopicQos** (p. 1397) values associated with the input XML QoS profile.*
- virtual **DDS_ReturnCode_t get_topic_qos_from_profile_w_topic_name** (**DDS_TopicQos** &qos, const char ∗library_name, const char ∗profile_name, const char ∗topic_name)=0

  *<<**extension**>> (p. 432) Gets the **DDS_TopicQos** (p. 1397) values associated with the input XML QoS profile while applying topic filters to the input topic name.*
- virtual **DDS_ReturnCode_t get_qos_profile_libraries** (struct **DDS_StringSeq** &library_names)=0

  *<<**extension**>> (p. 432) Gets the names of all XML QoS profile libraries associated with the **DDSDomainParticipant↩ Factory** (p. 1694)*
- virtual **DDS_ReturnCode_t get_qos_profiles** (struct **DDS_StringSeq** &profile_names, const char ∗library_↩ name)=0

  *<<**extension**>> (p. 432) Gets the names of all XML QoS profiles associated with the input XML QoS profile library.*
- virtual **DDSDomainParticipant** ∗ **create_participant** (**DDS_DomainId_t** domainId, const **DDS_Domain↩ ParticipantQos** &qos, **DDSDomainParticipantListener** ∗listener, **DDS_StatusMask** mask)=0

  *Creates a new **DDSDomainParticipant** (p. 1619) object.*
- virtual **DDSDomainParticipant** ∗ **create_participant_with_profile** (**DDS_DomainId_t** domainId, const char ∗library_name, const char ∗profile_name, **DDSDomainParticipantListener** ∗listener, **DDS_StatusMask** mask)=0

  *<<**extension**>> (p. 432) Creates a new **DDSDomainParticipant** (p. 1619) object using the **DDS_Domain↩ ParticipantQos** (p. 971) associated with the input XML QoS profile.*
- virtual **DDS_ReturnCode_t delete_participant** (**DDSDomainParticipant** ∗a_participant)=0

*Deletes an existing **DDSDomainParticipant** (p. 1619).*

- virtual **DDSDomainParticipant** * **lookup_participant** (**DDS_DomainId_t** domainId)=0

  *Locates an existing **DDSDomainParticipant** (p. 1619).*
- virtual **DDS_ReturnCode_t** **set_qos** (const **DDS_DomainParticipantFactoryQos** &qos)=0

  Sets the value for a participant factory QoS.
- virtual **DDS_ReturnCode_t** **get_qos** (**DDS_DomainParticipantFactoryQos** &qos)=0

  Gets the value for participant factory QoS.
- virtual **DDS_ReturnCode_t** **load_profiles** ()=0

  *<<**extension**>> (p. 432) Loads the XML QoS profiles.*
- virtual **DDS_ReturnCode_t** **reload_profiles** ()=0

  *<<**extension**>> (p. 432) Reloads the XML QoS profiles.*
- virtual **DDS_ReturnCode_t** **unload_profiles** ()=0

  *<<**extension**>> (p. 432) Unloads the XML QoS profiles.*
- virtual **DDS_ReturnCode_t** **unregister_thread** ()=0

  *<<**extension**>> (p. 432) Allows you to release thread-specific resources kept by RTI Connext on platforms that do not support automatic thread-specific storage (TSS) cleanup.*
- virtual const **DDS_TypeCode** * **get_typecode_from_config** (const char *type_name)=0

  *<<**extension**>> (p. 432) Gets a **DDS_TypeCode** (p. 1428) from a definition provided in an XML configuration file.*
- virtual **DDSDomainParticipant** * **create_participant_from_config** (const char *configuration_name)=0

  *<<**extension**>> (p. 432) Creates a **DDSDomainParticipant** (p. 1619) given its configuration name from a description provided in an XML configuration file.*
- virtual **DDSDomainParticipant** * **create_participant_from_config_w_params** (const char *configuration_↩ name, const **DDS_DomainParticipantConfigParams_t** &params)=0

  *<<**extension**>> (p. 432) Creates a **DDSDomainParticipant** (p. 1619) given its configuration name from a description provided in an XML configuration file and a set of parameters that allow changing some properties of such configuration.*
- virtual **DDSDomainParticipant** * **lookup_participant_by_name** (const char *participant_name)=0

  *<<**extension**>> (p. 432) Looks up a **DDSDomainParticipant** (p. 1619) by its entity name in the **DDSDomain↩ ParticipantFactory** (p. 1694).*
- virtual **DDS_ReturnCode_t** **register_type_support** (**DDSDomainParticipantFactory_RegisterType↩ Function** register_type_fcn, const char *type_name)=0

  *<<**extension**>> (p. 432) Registers a **DDSTypeSupport** (p. 1897) with the **DDSDomainParticipantFactory** (p. 1694) to enable automatic registration if the corresponding type, should it be needed by a **DDSDomainParticipant** (p. 1619).*
- virtual **DDS_ReturnCode_t** **get_participants** (DDSDomainParticipantSeq &participants)=0

  *<<**extension**>> (p. 432) Allows the application to access all the participants the DomainParticipantFactory has.*
- virtual **DDS_ReturnCode_t** **set_thread_factory** (**DDSThreadFactory** *thread_factory)=0

  *<<**extension**>> (p. 432) Sets a **DDSThreadFactory** (p. 1883) in the **DDSDomainParticipantFactory** (p. 1694) that will be used to create the internal threads of the DDS middleware.*

Static Public Member Functions

- static **DDSDomainParticipantFactory** * **get_instance** ()

  Gets the singleton instance of this class.
- static **DDS_ReturnCode_t** **finalize_instance** ()

  *<<**extension**>> (p. 432) Destroys the singleton instance of this class.*

9.254.1 Detailed Description

*<<**singleton**>> (p. 433) <<**interface**>> (p. 433)* Allows creation and destruction of **DDSDomainParticipant** (p. 1619) objects.

The sole purpose of this class is to allow the creation and destruction of **DDSDomainParticipant** (p. 1619) objects. This class itself is a *<<**singleton**>> (p. 433)*, and accessed via the **get_instance()** (p. 1697) method, and destroyed with **finalize_instance()** (p. 1697) method.

A single application can participate in multiple domains by instantiating multiple **DDSDomainParticipant** (p. 1619) objects.

An application may even instantiate multiple participants in the same domain. Participants in the same domain exchange data in the same way regardless of whether they are in the same application or different applications or on the same node or different nodes; their location is transparent.

There are two important caveats:

- When there are multiple participants on the same node (in the same application or different applications) in the same domain, the application(s) must make sure that the participants do not try to bind to the same port numbers. You must disambiguate between the participants by setting a participant ID for each participant (**DDS_Wire↩ ProtocolQosPolicy::participant_id** (p. 1514)). The port numbers used by a participant are calculated based on both the participant index and the domain ID, so if all participants on the same node have different participant indexes, they can coexist in the same domain.
- You cannot mix entities from different participants. For example, you cannot delete a topic on a different participant than you created it from, and you cannot ask a subscriber to create a reader for a topic created from a participant different than the subscriber's own participant. (Note that it is permissable for an application built on top of RTI Connext to know about entities from different participants. For example, an application could keep references to a reader from one domain and a writer from another and then bridge the domains by writing the data received in the reader callback.)

See also

DDSDomainParticipant (p. 1619)

9.254.2 Member Function Documentation

9.254.2.1 get_instance()

```
static DDSDomainParticipantFactory * DDSDomainParticipantFactory::get_instance ( ) [static]
```

Gets the singleton instance of this class.

DDSTheParticipantFactory (p. 43) can be used as an alias for the singleton factory returned by this operation.

Returns

The singleton **DDSDomainParticipantFactory** (p. 1694) instance.

See also

DDSTheParticipantFactory (p. 43)

9.254.2.2 finalize_instance()

```
static DDS_ReturnCode_t DDSDomainParticipantFactory::finalize_instance ( ) [static]
```

<<***extension***>> *(*p. *432)* Destroys the singleton instance of this class.

Only necessary to explicitly reclaim resources used by the participant factory singleton. Note that on many OSs, these resources are automatically reclaimed by the OS when the program terminates. However, some memory-check tools still flag these as unreclaimed. So this method provides a way to clean up memory used by the participant factory.

Precondition

All participants created from the factory have been deleted.

Postcondition

All resources belonging to the factory have been reclaimed. Another call to **DDSDomainParticipantFactory::get_instance** (p. 1697) will return a new lifecycle of the singleton.

Exceptions

One	of the **Standard Return Codes** (p. 532), or **DDS_RETCODE_PRECONDITION_NOT_MET** (p. 532)

See also

DDSTheParticipantFactory (p. 43)

Examples

HelloWorld_publisher.cxx, and **HelloWorld_subscriber.cxx**.

9.254.2.3 set_default_participant_qos()

```
virtual DDS_ReturnCode_t DDSDomainParticipantFactory::set_default_participant_qos (
            const DDS_DomainParticipantQos & qos ) [pure virtual]
```

Sets the default **DDS_DomainParticipantQos** (p. 971) values for this domain participant factory.

This method may potentially allocate memory depending on the sequences contained in some QoS policies.

Parameters

qos	<<***inout***>> *(p. 433)* Qos to be filled up. The special value **DDS_PARTICIPANT_QOS_DEFAULT** (p. 50) may be passed as `qos` to indicate that the default QoS should be reset back to the initial values the factory would used if **DDSDomainParticipantFactory::set_default_participant_qos** (p. 1698) had never been called.

Exceptions

One	of the **Standard Return Codes** (p. 532)

See also

DDS_PARTICIPANT_QOS_DEFAULT (p. 50)

DDSDomainParticipantFactory::create_participant (p. 1710)

9.254.2.4 set_default_participant_qos_with_profile()

```
virtual  DDS_ReturnCode_t DDSDomainParticipantFactory::set_default_participant_qos_with_profile (
             const char * library_name,
             const char * profile_name )  [pure virtual]
```

<<***extension***>> *(p. 432)* Sets the default **DDS_DomainParticipantQos** (p. 971) values for this domain participant factory based on the input XML QoS profile.

This method may potentially allocate memory depending on the sequences contained in some QoS policies.

This default value will be used for newly created **DDSDomainParticipant** (p. 1619) if **DDS_PARTICIPANT_QOS_DEFAULT** (p. 50) is specified as the `qos` parameter when **DDSDomainParticipantFactory::create_participant** (p. 1710) is called.

Precondition

The **DDS_DomainParticipantQos** (p. 971) contained in the specified XML QoS profile must be consistent, or else the operation will have no effect and fail with **DDS_RETCODE_INCONSISTENT_POLICY** (p. 533)

MT Safety:

UNSAFE. It is not safe to retrieve the default QoS value from a domain participant factory while another thread may be simultaneously calling **DDSDomainParticipantFactory::set_default_participant_qos** (p. 1698)

Parameters

library_name	<<***in***>> *(p. 433)* Library name containing the XML QoS profile. If library_name is null RTI Connext will use the default library (see **DDSDomainParticipantFactory::set_default_library** (p. 1701)).
profile_name	<<***in***>> *(p. 433)* XML QoS Profile name. If profile_name is null RTI Connext will use the default profile (see **DDSDomainParticipantFactory::set_default_profile** (p. 1702)).

If the input profile cannot be found the method fails with **DDS_RETCODE_ERROR** (p. 532).

Exceptions

One	of the **Standard Return Codes** (p. 532), or **DDS_RETCODE_INCONSISTENT_POLICY** (p. 533)

See also

DDS_PARTICIPANT_QOS_DEFAULT (p. 50)

DDSDomainParticipantFactory::create_participant_with_profile (p. 1711)

9.254.2.5 get_default_participant_qos()

```
virtual DDS_ReturnCode_t DDSDomainParticipantFactory::get_default_participant_qos (
            DDS_DomainParticipantQos & qos ) [pure virtual]
```

Initializes the **DDS_DomainParticipantQos** (p. 971) instance with default values.

The retrieved `qos` will match the set of values specified on the last successful call to **DDSDomainParticipantFactory::set_default_participant_qos** (p. 1698), or **DDSDomainParticipantFactory::set_default_participant_qos_with_profile** (p. 1699), or else, if the call was never made, the default values listed in **DDS_DomainParticipantQos** (p. 971).

This method may potentially allocate memory depending on the sequences contained in some QoS policies.

Parameters

qos	<<***out***>> *(p. 433)* the domain participant's QoS

Exceptions

One	of the **Standard Return Codes** (p. 532)

See also

DDS_PARTICIPANT_QOS_DEFAULT (p. 50)

DDSDomainParticipantFactory::create_participant (p. 1710)

9.254.2.6 set_default_library()

```
virtual  DDS_ReturnCode_t DDSDomainParticipantFactory::set_default_library (
            const char * library_name )  [pure virtual]
```

<<***extension***>> *(p. 432)* Sets the default XML library for a **DDSDomainParticipantFactory** (p. 1694).

Any API requiring a library_name as a parameter can use NULL to refer to the default library set with this function.

Note: if the library set with this function no longer exists after reloading the QoS profiles (for example, by changing **DDS_DomainParticipantFactoryQos::profile** (p. 969)) the default library will be set to the last library containing a profile with the attribute `is_default_qos=true` or NULL no such library exists.

See also

DDSDomainParticipantFactory::set_default_profile (p. 1702) for more information.

Parameters

library_name	<<***in***>> *(p. 433)* Library name. If library_name is NULL any previous default is unset.

Exceptions

One	of the **Standard Return Codes** (p. 532)

See also

DDSDomainParticipantFactory::get_default_library (p. 1701)

9.254.2.7 get_default_library()

```
virtual const char * DDSDomainParticipantFactory::get_default_library ( )  [pure virtual]
```

<<***extension***>> *(p. 432)* Gets the default XML library associated with a **DDSDomainParticipantFactory** (p. 1694).

Returns

The returned library name is determined as follows:

- If it was previously set with **DDSDomainParticipantFactory::set_default_library** (p. 1701), this function returns that library name
- Otherwise, if one or more profiles have the XML attribute `is_default_qos="true"`, this function returns the library where the last one is contained.
- Otherwise, this function returns NULL.

See also

DDSDomainParticipantFactory::set_default_library (p. 1701)

9.254.2.8 set_default_profile()

```
virtual  DDS_ReturnCode_t DDSDomainParticipantFactory::set_default_profile (
             const char * library_name,
             const char * profile_name )  [pure virtual]
```

<<***extension***>> *(p. 432)* Sets the default XML profile for a **DDSDomainParticipantFactory** (p. 1694).

This method specifies the profile that will be used as the default the next time a default DomainParticipantFactory profile is needed during a call to a DomainParticipantFactory method. When calling a **DDSDomainParticipantFactory** (p. 1694) method that requires a `profile_name` parameter, you can use NULL to refer to the default profile. (This same information applies to setting a default library.)

This method does not set the default QoS for newly created DomainParticipants; for this functionality, use **DDSDomainParticipantFactory::set_default_participant_qos_with_profile** (p. 1699) (you may pass in NULL after having called **set_default_profile()** (p. 1702)).

Note: if the profile set with this function no longer exists after reloading the QoS profiles (for example, by changing **DDS_DomainParticipantFactoryQos::profile** (p. 969)) the default profile will be set to the last one marked with the attribute `is_default_qos=true` or NULL no such profile exists.

Parameters

library_name	<<***in***>> *(p. 433)* The library name containing the profile.
profile_name	<<***in***>> *(p. 433)* The profile name. If profile_name is NULL any previous default is unset.

Exceptions

One	of the **Standard Return Codes** (p. 532)

See also

DDSDomainParticipantFactory::get_default_profile (p. 1703)

DDSDomainParticipantFactory::get_default_profile_library (p. 1703)

9.254.2.9 get_default_profile()

```
virtual const char * DDSDomainParticipantFactory::get_default_profile ( )  [pure virtual]
```

<<***extension***>> *(p. 432)* Gets the default XML profile associated with a **DDSDomainParticipantFactory** (p. 1694).

Returns

The returned profile name is determined as follows:

- If it was previously set with **DDSDomainParticipantFactory::set_default_profile** (p. 1702), this function returns that profile name
- Otherwise, if one or more profiles have the XML attribute `is_default_qos="true"`, this function returns the name of one of them
- Otherwise, this function returns NULL.

See also

DDSDomainParticipantFactory::set_default_profile (p. 1702)

9.254.2.10 get_default_profile_library()

```
virtual const char * DDSDomainParticipantFactory::get_default_profile_library ( )  [pure virtual]
```

<<***extension***>> *(p. 432)* Gets the library where the default XML profile is contained for a **DDSDomainParticipant↩ Factory** (p. 1694).

The default profile library is automatically set when **DDSDomainParticipantFactory::set_default_profile** (p. 1702) is called.

This library can be different than the **DDSDomainParticipantFactory** (p. 1694) default library (see **DDSDomain↩ ParticipantFactory::get_default_library** (p. 1701)).

Returns

The default profile library for the profile returned by **DDSDomainParticipantFactory::get_default_profile** (p. 1703), or NULL if that profile is NULL.

See also

DDSDomainParticipantFactory::get_default_profile (p. 1703)

9.254.2.11 get_participant_factory_qos_from_profile()

```
virtual DDS_ReturnCode_t DDSDomainParticipantFactory::get_participant_factory_qos_from_profile (
            DDS_DomainParticipantFactoryQos & qos,
            const char * library_name,
            const char * profile_name ) [pure virtual]
```

<<***extension***>> *(p. 432)* Gets the **DDS_DomainParticipantFactoryQos** (p. 967) values associated with the input XML QoS profile.

Parameters

qos	<<***out***>> *(p. 433)* Qos to be filled up. Cannot be NULL.
library_name	<<***in***>> *(p. 433)* Library name containing the XML QoS profile. If library_name is null RTI Connext will use the default library (see **DDSDomainParticipantFactory::set_default_library** (p. 1701)).
profile_name	<<***in***>> *(p. 433)* XML QoS Profile name. If profile_name is null RTI Connext will use the default profile (see **DDSDomainParticipantFactory::set_default_profile** (p. 1702)).

If the input profile cannot be found, the method fails with **DDS_RETCODE_ERROR** (p. 532).

Exceptions

One	of the **Standard Return Codes** (p. 532)

9.254.2.12 get_participant_qos_from_profile()

```
virtual DDS_ReturnCode_t DDSDomainParticipantFactory::get_participant_qos_from_profile (
            DDS_DomainParticipantQos & qos,
            const char * library_name,
            const char * profile_name ) [pure virtual]
```

<<***extension***>> *(p. 432)* Gets the **DDS_DomainParticipantQos** (p. 971) values associated with the input XML QoS profile.

Parameters

qos	<<***out***>> *(p. 433)* Qos to be filled up. Cannot be NULL.
library_name	<<***in***>> *(p. 433)* Library name containing the XML QoS profile. If library_name is null RTI Connext will use the default library (see **DDSDomainParticipantFactory::set_default_library** (p. 1701)).
profile_name	<<***in***>> *(p. 433)* XML QoS Profile name. If profile_name is null RTI Connext will use the default profile (see **DDSDomainParticipantFactory::set_default_profile** (p. 1702)).

If the input profile cannot be found, the method fails with **DDS_RETCODE_ERROR** (p. 532).

Exceptions

One	of the **Standard Return Codes** (p. 532)

9.254.2.13 get_publisher_qos_from_profile()

```
virtual DDS_ReturnCode_t DDSDomainParticipantFactory::get_publisher_qos_from_profile (
            DDS_PublisherQos & qos,
            const char * library_name,
            const char * profile_name ) [pure virtual]
```

<<***extension***>> *(p. 432)* Gets the **DDS_PublisherQos** (p. 1276) values associated with the input XML QoS profile.

Parameters

qos	<<***out***>> *(p. 433)* Qos to be filled up. Cannot be NULL.
library_name	<<***in***>> *(p. 433)* Library name containing the XML QoS profile. If library_name is null RTI Connext will use the default library (see **DDSDomainParticipantFactory::set_default_library** (p. 1701)).
profile_name	<<***in***>> *(p. 433)* XML QoS Profile name. If profile_name is null RTI Connext will use the default profile (see **DDSDomainParticipantFactory::set_default_profile** (p. 1702)).

If the input profile cannot be found, the method fails with **DDS_RETCODE_ERROR** (p. 532).

Exceptions

One	of the **Standard Return Codes** (p. 532)

9.254.2.14 get_subscriber_qos_from_profile()

```
virtual DDS_ReturnCode_t DDSDomainParticipantFactory::get_subscriber_qos_from_profile (
            DDS_SubscriberQos & qos,
            const char * library_name,
            const char * profile_name ) [pure virtual]
```

<<***extension***>> *(p. 432)* Gets the **DDS_SubscriberQos** (p. 1367) values associated with the input XML QoS profile.

Parameters

qos	<<***out***>> *(p. 433)* Qos to be filled up. Cannot be NULL.

Parameters

library_name	<<***in***>> *(p. 433)* Library name containing the XML QoS profile. If library_name is null RTI Connext will use the default library (see **DDSDomainParticipantFactory::set_default_library** (p. 1701)).
profile_name	<<***in***>> *(p. 433)* XML QoS Profile name. If profile_name is null RTI Connext will use the default profile (see **DDSDomainParticipantFactory::set_default_profile** (p. 1702)).

If the input profile cannot be found, the method fails with **DDS_RETCODE_ERROR** (p. 532).

Exceptions

One	of the **Standard Return Codes** (p. 532)

9.254.2.15 get_datawriter_qos_from_profile()

```
virtual DDS_ReturnCode_t DDSDomainParticipantFactory::get_datawriter_qos_from_profile (
            DDS_DataWriterQos & qos,
            const char * library_name,
            const char * profile_name ) [pure virtual]
```

<<***extension***>> *(p. 432)* Gets the **DDS_DataWriterQos** (p. 909) values associated with the input XML QoS profile.

Parameters

qos	<<***out***>> *(p. 433)* Qos to be filled up. Cannot be NULL.
library_name	<<***in***>> *(p. 433)* Library name containing the XML QoS profile. If library_name is null RTI Connext will use the default library (see **DDSDomainParticipantFactory::set_default_library** (p. 1701)).
profile_name	<<***in***>> *(p. 433)* XML QoS Profile name. If profile_name is null RTI Connext will use the default profile (see **DDSDomainParticipantFactory::set_default_profile** (p. 1702)).

If the input profile cannot be found, the method fails with **DDS_RETCODE_ERROR** (p. 532).

Exceptions

One	of the **Standard Return Codes** (p. 532)

9.254.2.16 get_datawriter_qos_from_profile_w_topic_name()

```
virtual DDS_ReturnCode_t DDSDomainParticipantFactory::get_datawriter_qos_from_profile_w_topic_↩
name (
```

```
 DDS_DataWriterQos & qos,
const char * library_name,
const char * profile_name,
const char * topic_name )  [pure virtual]
```

<<***extension***>> *(p. 432)* Gets the **DDS_DataWriterQos** (p. 909) values associated with the input XML QoS profile while applying topic filters to the input topic name.

Parameters

qos	<<***out***>> *(p. 433)* Qos to be filled up. Cannot be NULL.
library_name	<<***in***>> *(p. 433)* Library name containing the XML QoS profile. If library_name is null RTI Connext will use the default library (see **DDSDomainParticipantFactory::set_default_library** (p. 1701)).
profile_name	<<***in***>> *(p. 433)* XML QoS Profile name. If profile_name is null RTI Connext will use the default profile (see **DDSDomainParticipantFactory::set_default_profile** (p. 1702)).
topic_name	<<***in***>> *(p. 433)* Topic name that will be evaluated against the topic_filter attribute in the XML QoS profile. If topic_name is null, RTI Connext will match only QoSs without explicit topic_filter expressions.

If the input profile cannot be found, the method fails with **DDS_RETCODE_ERROR** (p. 532).

Exceptions

One	of the **Standard Return Codes** (p. 532)

9.254.2.17 get_datareader_qos_from_profile()

```
virtual  DDS_ReturnCode_t DDSDomainParticipantFactory::get_datareader_qos_from_profile (
             DDS_DataReaderQos & qos,
            const char * library_name,
            const char * profile_name )  [pure virtual]
```

<<***extension***>> *(p. 432)* Gets the **DDS_DataReaderQos** (p. 858) values associated with the input XML QoS profile.

Parameters

qos	<<***out***>> *(p. 433)* Qos to be filled up. Cannot be NULL.
library_name	<<***in***>> *(p. 433)* Library name containing the XML QoS profile. If library_name is null RTI Connext will use the default library (see **DDSDomainParticipantFactory::set_default_library** (p. 1701)).
profile_name	<<***in***>> *(p. 433)* XML QoS Profile name. If profile_name is null RTI Connext will use the default profile (see **DDSDomainParticipantFactory::set_default_profile** (p. 1702)).

If the input profile cannot be found, the method fails with **DDS_RETCODE_ERROR** (p. 532).

Exceptions

One	of the **Standard Return Codes** (p. 532)

9.254.2.18 get_datareader_qos_from_profile_w_topic_name()

```
virtual DDS_ReturnCode_t DDSDomainParticipantFactory::get_datareader_qos_from_profile_w_topic_↩
name (
            DDS_DataReaderQos & qos,
            const char * library_name,
            const char * profile_name,
            const char * topic_name ) [pure virtual]
```

<<***extension***>> *(p. 432)* Gets the **DDS_DataReaderQos** (p. 858) values associated with the input XML QoS profile while applying topic filters to the input topic name.

Parameters

qos	<<***out***>> *(p. 433)* Qos to be filled up. Cannot be NULL.
library_name	<<***in***>> *(p. 433)* Library name containing the XML QoS profile. If library_name is null RTI Connext will use the default library (see **DDSDomainParticipantFactory::set_default_library** (p. 1701)).
profile_name	<<***in***>> *(p. 433)* XML QoS Profile name. If profile_name is null RTI Connext will use the default profile (see **DDSDomainParticipantFactory::set_default_profile** (p. 1702)).
topic_name	<<***in***>> *(p. 433)* Topic name that will be evaluated against the topic_filter attribute in the XML QoS profile. If topic_name is null, RTI Connext will match only QoSs without explicit topic_filter expressions.

If the input profile cannot be found, the method fails with **DDS_RETCODE_ERROR** (p. 532).

Exceptions

One	of the **Standard Return Codes** (p. 532)

9.254.2.19 get_topic_qos_from_profile()

```
virtual DDS_ReturnCode_t DDSDomainParticipantFactory::get_topic_qos_from_profile (
            DDS_TopicQos & qos,
            const char * library_name,
            const char * profile_name ) [pure virtual]
```

<<***extension***>> *(p. 432)* Gets the **DDS_TopicQos** (p. 1397) values associated with the input XML QoS profile.

Parameters

qos	<<***out***>> *(p. 433)* Qos to be filled up. Cannot be NULL.
library_name	<<***in***>> *(p. 433)* Library name containing the XML QoS profile. If library_name is null RTI Connext will use the default library (see **DDSDomainParticipantFactory::set_default_library** (p. 1701)).
profile_name	<<***in***>> *(p. 433)* XML QoS Profile name. If profile_name is null RTI Connext will use the default profile (see **DDSDomainParticipantFactory::set_default_profile** (p. 1702)).

If the input profile cannot be found, the method fails with **DDS_RETCODE_ERROR** (p. 532).

Exceptions

One	of the **Standard Return Codes** (p. 532)

9.254.2.20 get_topic_qos_from_profile_w_topic_name()

```
virtual DDS_ReturnCode_t DDSDomainParticipantFactory::get_topic_qos_from_profile_w_topic_name (
             DDS_TopicQos & qos,
            const char * library_name,
            const char * profile_name,
            const char * topic_name )  [pure virtual]
```

<<***extension***>> *(p. 432)* Gets the **DDS_TopicQos** (p. 1397) values associated with the input XML QoS profile while applying topic filters to the input topic name.

Parameters

qos	<<***out***>> *(p. 433)* Qos to be filled up. Cannot be NULL.
library_name	<<***in***>> *(p. 433)* Library name containing the XML QoS profile. If library_name is null RTI Connext will use the default library (see **DDSDomainParticipantFactory::set_default_library** (p. 1701)).
profile_name	<<***in***>> *(p. 433)* XML QoS Profile name. If profile_name is null RTI Connext will use the default profile (see **DDSDomainParticipantFactory::set_default_profile** (p. 1702)).
topic_name	<<***in***>> *(p. 433)* Topic name that will be evaluated against the topic_filter attribute in the XML QoS profile. If topic_name is null, RTI Connext will match only QoSs without explicit topic_filter expressions.

If the input profile cannot be found, the method fails with **DDS_RETCODE_ERROR** (p. 532).

Exceptions

One	of the **Standard Return Codes** (p. 532)

9.254.2.21 get_qos_profile_libraries()

```
virtual DDS_ReturnCode_t DDSDomainParticipantFactory::get_qos_profile_libraries (
            struct DDS_StringSeq & library_names ) [pure virtual]
```

<<***extension***>> *(p. 432)* Gets the names of all XML QoS profile libraries associated with the **DDSDomain↩ ParticipantFactory** (p. 1694)

Parameters

library_names	<<***out***>> *(p. 433)* **DDS_StringSeq** (p. 1364) to be filled with names of XML QoS profile libraries. Cannot be NULL.

9.254.2.22 get_qos_profiles()

```
virtual DDS_ReturnCode_t DDSDomainParticipantFactory::get_qos_profiles (
            struct DDS_StringSeq & profile_names,
            const char * library_name ) [pure virtual]
```

<<***extension***>> *(p. 432)* Gets the names of all XML QoS profiles associated with the input XML QoS profile library.

Parameters

profile_names	<<***out***>> *(p. 433)* **DDS_StringSeq** (p. 1364) to be filled with names of XML QoS profiles. Cannot be NULL.
library_name	<<***in***>> *(p. 433)* Library name containing the XML QoS profile. If library_name is null RTI Connext will use the default library (see **DDSDomainParticipantFactory::set_default_library** (p. 1701)).

9.254.2.23 create_participant()

```
virtual DDSDomainParticipant * DDSDomainParticipantFactory::create_participant (
             DDS_DomainId_t domainId,
            const DDS_DomainParticipantQos & qos,
             DDSDomainParticipantListener * listener,
             DDS_StatusMask mask ) [pure virtual]
```

Creates a new **DDSDomainParticipant** (p. 1619) object.

Precondition

The specified QoS policies must be consistent or the operation will fail and no **DDSDomainParticipant** (p. 1619) will be created.

If you want to create multiple participants on a given host in the same domain, make sure each one has a different participant index (set in the **DDS_WireProtocolQosPolicy** (p. 1511)). This in turn will ensure each participant uses a different port number (since the unicast port numbers are calculated from the participant index and the domain ID).

Note that if there is a single participant per host in a given domain, the participant index can be left at the default value (-1).

MT Safety:

UNSAFE. On VxWorks, when enabling the RTI Security Plugins, it is not safe for multiple threads to attempt to create a **DDSDomainParticipant** (p. 1619) when there are no other DomainParticipants in the **DDSDomainParticipantFactory** (p. 1694).

Parameters

domain_id	<<***in***>> *(p. 433)* ID of the domain that the application intends to join. **[range]** [>=0], and does not violate guidelines stated in **DDS_RtpsWellKnownPorts_t** (p. 1338).
qos	<<***in***>> *(p. 433)* the DomainParticipant's QoS. The special value **DDS_PARTICIPANT_QOS_DEFAULT** (p. 50) can be used to indicate that the **DDSDomainParticipant** (p. 1619) should be created with the default **DDS_DomainParticipantQos** (p. 971) set in the **DDSDomainParticipantFactory** (p. 1694).
listener	<<***in***>> *(p. 433)* the domain participant's listener.
mask	<<***in***>> *(p. 433)*. Changes of communication status to be invoked on the listener. See **DDS_StatusMask** (p. 537).

Returns

domain participant or NULL on failure

See also

Specifying QoS on entities (p. 555) for information on setting QoS before entity creation

DDS_DomainParticipantQos (p. 971) for rules on consistency among QoS

DDS_PARTICIPANT_QOS_DEFAULT (p. 50)

NDDS_DISCOVERY_PEERS (p. 662)

DDSDomainParticipantFactory::create_participant_with_profile (p. 1711)

DDSDomainParticipantFactory::get_default_participant_qos (p. 1700)

DDSDomainParticipant::set_listener (p. 1682)

9.254.2.24 create_participant_with_profile()

```
virtual DDSDomainParticipant * DDSDomainParticipantFactory::create_participant_with_profile (
            DDS_DomainId_t domainId,
            const char * library_name,
            const char * profile_name,
            DDSDomainParticipantListener * listener,
            DDS_StatusMask mask ) [pure virtual]
```

<<***extension***>> *(p. 432)* Creates a new **DDSDomainParticipant** (p. 1619) object using the **DDS_Domain↩ ParticipantQos** (p. 971) associated with the input XML QoS profile.

Precondition

The **DDS_DomainParticipantQos** (p. 971) in the input profile must be consistent, or the operation will fail and no **DDSDomainParticipant** (p. 1619) will be created.

If you want to create multiple participants on a given host in the same domain, make sure each one has a different participant index (set in the **DDS_WireProtocolQosPolicy** (p. 1511)). This in turn will ensure each participant uses a different port number (since the unicast port numbers are calculated from the participant index and the domain ID).

Note that if there is a single participant per host in a given domain, the participant index can be left at the default value (-1).

MT Safety:

UNSAFE. On VxWorks, when enabling the RTI Security Plugins, it is not safe for multiple threads to attempt to create a **DDSDomainParticipant** (p. 1619) when there are no other DomainParticipants in the **DDSDomain↩ ParticipantFactory** (p. 1694).

Parameters

domainId	<<***in***>> *(p. 433)* ID of the domain that the application intends to join. **[range]** [>=0], and does not violate guidelines stated in **DDS_RtpsWellKnownPorts_t** (p. 1338).
library_name	<<***in***>> *(p. 433)* Library name containing the XML QoS profile. If library_name is null RTI Connext will use the default library (see **DDSDomainParticipantFactory::set_default_library** (p. 1701)).
profile_name	<<***in***>> *(p. 433)* XML QoS Profile name. If profile_name is null RTI Connext will use the default profile (see **DDSDomainParticipantFactory::set_default_profile** (p. 1702)).
listener	<<***in***>> *(p. 433)* the DomainParticipant's listener.
mask	<<***in***>> *(p. 433)*. Changes of communication status to be invoked on the listener. See **DDS_StatusMask** (p. 537).

Returns

domain participant or NULL on failure

See also

Specifying QoS on entities (p. 555) for information on setting QoS before entity creation

DDS_DomainParticipantQos (p. 971) for rules on consistency among QoS

DDS_PARTICIPANT_QOS_DEFAULT (p. 50)

NDDS_DISCOVERY_PEERS (p. 662)

DDSDomainParticipantFactory::create_participant() (p. 1710)

DDSDomainParticipantFactory::get_default_participant_qos() (p. 1700)

DDSDomainParticipant::set_listener() (p. 1682)

9.254.2.25 delete_participant()

```
virtual DDS_ReturnCode_t DDSDomainParticipantFactory::delete_participant (
            DDSDomainParticipant * a_participant ) [pure virtual]
```

Deletes an existing **DDSDomainParticipant** (p. 1619).

Precondition

All domain entities belonging to the participant must have already been deleted. Otherwise it fails with the error **DDS_RETCODE_PRECONDITION_NOT_MET** (p. 532).

Postcondition

Listener installed on the **DDSDomainParticipant** (p. 1619) will not be called after this method returns successfully.

MT Safety:

UNSAFE. It is not safe to delete an entity while another thread may be simultaneously calling an API that uses the entity. On VxWorks, when enabling the RTI Security Plugins, if two threads attempt to delete the last two DomainParticipants of the **DDSDomainParticipantFactory** (p. 1694), it is possible for memory to be leaked.

Parameters

a_participant	<<***in***>> *(p. 433)* **DDSDomainParticipant** (p. 1619) to be deleted.

Exceptions

One	of the **Standard Return Codes** (p. 532), or **DDS_RETCODE_PRECONDITION_NOT_MET** (p. 532).

9.254.2.26 lookup_participant()

```
virtual DDSDomainParticipant * DDSDomainParticipantFactory::lookup_participant (
            DDS_DomainId_t domainId ) [pure virtual]
```

Locates an existing **DDSDomainParticipant** (p. 1619).

If no such **DDSDomainParticipant** (p. 1619) exists, the operation will return NULL value.

If multiple **DDSDomainParticipant** (p. 1619) entities belonging to that domainId exist, then the operation will return one of them. It is not specified which one.

Parameters

domain↩ Id	<<***in***>> *(p. 433)* ID of the domain participant to lookup.

Returns

domain participant if it exists, or NULL

9.254.2.27 set_qos()

```
virtual DDS_ReturnCode_t DDSDomainParticipantFactory::set_qos (
            const DDS_DomainParticipantFactoryQos & qos ) [pure virtual]
```

Sets the value for a participant factory QoS.

The **DDS_DomainParticipantFactoryQos::entity_factory** (p. 969) can be changed. The other policies are immutable.

Note that despite having QoS, the **DDSDomainParticipantFactory** (p. 1694) is not an **DDSEntity** (p. 1731).

Parameters

qos	<<***in***>> *(p. 433)* Set of policies to be applied to **DDSDomainParticipantFactory** (p. 1694). Policies must be consistent. Immutable policies can only be changed before calling any other RTI Connext methods except for **DDSDomainParticipantFactory::get_qos** (p. 1715)

Exceptions

One	of the **Standard Return Codes** (p. 532), **DDS_RETCODE_IMMUTABLE_POLICY** (p. 533) if immutable policy is changed, or **DDS_RETCODE_INCONSISTENT_POLICY** (p. 533) if policies are inconsistent

See also

DDS_DomainParticipantFactoryQos (p. 967) for rules on consistency among QoS

9.254.2.28 get_qos()

```
virtual DDS_ReturnCode_t DDSDomainParticipantFactory::get_qos (
            DDS_DomainParticipantFactoryQos & qos ) [pure virtual]
```

Gets the value for participant factory QoS.

Parameters

qos	<<***inout***>> *(p. 433)* QoS to be filled up.

Exceptions

One	of the **Standard Return Codes** (p. 532)

9.254.2.29 load_profiles()

```
virtual DDS_ReturnCode_t DDSDomainParticipantFactory::load_profiles ( ) [pure virtual]
```

<<***extension***>> *(p. 432)* Loads the XML QoS profiles.

The XML QoS profiles are loaded implicitly after the first **DDSDomainParticipant** (p. 1619) is created or explicitly, after a call to this method.

This has the same effect as **DDSDomainParticipantFactory::reload_profiles()** (p. 1715).

Exceptions

One	of the **Standard Return Codes** (p. 532)

See also

DDS_ProfileQosPolicy (p. 1257)

9.254.2.30 reload_profiles()

```
virtual DDS_ReturnCode_t DDSDomainParticipantFactory::reload_profiles ( ) [pure virtual]
```

<<***extension***>> *(p. 432)* Reloads the XML QoS profiles.

The XML QoS profiles are loaded implicitly after the first **DDSDomainParticipant** (p. 1619) is created or explicitly, after a call to this method.

This has the same effect as **DDSDomainParticipantFactory::load_profiles()** (p. 1715).

Exceptions

One	of the **Standard Return Codes** (p. 532)

See also

DDS_ProfileQosPolicy (p. 1257)

9.254.2.31 unload_profiles()

```
virtual DDS_ReturnCode_t DDSDomainParticipantFactory::unload_profiles ( ) [pure virtual]
```

<<***extension***>> *(p. 432)* Unloads the XML QoS profiles.

The resources associated with the XML QoS profiles are freed. Any reference to the profiles after calling this method will fail with an error.

Exceptions

One	of the **Standard Return Codes** (p. 532)

See also

DDS_ProfileQosPolicy (p. 1257)

9.254.2.32 unregister_thread()

```
virtual DDS_ReturnCode_t DDSDomainParticipantFactory::unregister_thread ( ) [pure virtual]
```

<<***extension***>> *(p. 432)* Allows you to release thread-specific resources kept by RTI Connext on platforms that do not support automatic thread-specific storage (TSS) cleanup.

Refer to the Platform Notes to see if your platform supports automatic TSS cleanup. It will be mentioned in the 'Thread Configuration' sub-section of the platform you are using.

On platforms that do not support automatic TSS cleanup, you should call this function right before exiting a thread where any DDS API was used. It is safe to call this function even if no DDS API was used, in cases where you are unsure if a DDS API was called or not. It is also safe, although unnecessary, to call this API on platforms that support automatic TSS cleanup.

RTI Connext stores some resources in TSS. For internal threads, these resources are cleaned up by the middleware when the threads exit. For user threads, calling this API will free all the resources related to the specific thread.

The best approach is to call the function during the thread deletion after all the DDS APIs have been called.

See Unregistering Threads, in the Core Libraries User's Manual, for the equivalent API call in each of the supported language bindings.

Exceptions

One	of the **Standard Return Codes** (p. 532)

9.254.2.33 get_typecode_from_config()

```
virtual const  DDS_TypeCode * DDSDomainParticipantFactory::get_typecode_from_config (
            const char * type_name )  [pure virtual]
```

<<***extension***>> *(p. 432)* Gets a **DDS_TypeCode** (p. 1428) from a definition provided in an XML configuration file.

Parameters

type_name	<<***in***>> *(p. 433)* Name of the type in the configuration file.

Returns

The **DDS_TypeCode** (p. 1428) or NULL if a type with that name doesn't exist or an error occurs.

9.254.2.34 create_participant_from_config()

```
virtual  DDSDomainParticipant * DDSDomainParticipantFactory::create_participant_from_config (
            const char * configuration_name )  [pure virtual]
```

<<***extension***>> *(p. 432)* Creates a **DDSDomainParticipant** (p. 1619) given its configuration name from a description provided in an XML configuration file.

This operation creates a **DDSDomainParticipant** (p. 1619) registering all the necessary data types and creating all the contained entities (**DDSTopic** (p. 1885), **DDSPublisher** (p. 1818), **DDSSubscriber** (p. 1860), **DDSDataWriter** (p. 1589), **DDSDataReader** (p. 1556)) from a description given in an XML configuration file.

The configuration name is the fully qualified name of the XML participant object, consisting of the name of the participant library plus the name of participant configuration.

For example the name "MyParticipantLibrary::PublicationParticipant" can be used to create the domain participant from the description in an XML file with contents shown in the snippet below:

```
<participant_library name="MyParticipantLibrary">

    <domain_participant name="PublicationParticipant" domain_ref="MyDomainLibrary::HelloWorldDomain">

        <publisher name="MyPublisher">

            <data_writer name="HelloWorldWriter" topic_ref="HelloWorldTopic"/>

        </publisher>

    </domain_participant>

</participant_library>
```

The entities belonging to the newly created **DDSDomainParticipant** (p. 1619) can be retrieved with the help of lookup operations such as: **DDSDomainParticipant::lookup_datareader_by_name** (p. 1692).

This operation is equivalent to call **DDSDomainParticipantFactory::create_participant_from_config_w_params** (p. 1718) passing **DDS_PARTICIPANT_CONFIG_PARAMS_DEFAULT** (p. 50) as parameters.

MT Safety:

UNSAFE. On VxWorks, when enabling the RTI Security Plugins, it is not safe for multiple threads to attempt to create a **DDSDomainParticipant** (p. 1619) when there are no other DomainParticipants in the **DDSDomain↩ ParticipantFactory** (p. 1694).

Parameters

configuration_name	*<<**in**>> (p. 433)* Name of the participant configuration in the XML file.

Returns

The created **DDSDomainParticipant** (p. 1619) or NULL on error.

See also

DDSDomainParticipantFactory::lookup_participant_by_name (p. 1719)

DDSDomainParticipant::lookup_topicdescription (p. 1658)

DDSDomainParticipant::lookup_publisher_by_name (p. 1690)

DDSDomainParticipant::lookup_subscriber_by_name (p. 1690)

DDSDomainParticipant::lookup_datareader_by_name (p. 1692)

DDSDomainParticipant::lookup_datawriter_by_name (p. 1691)

DDSPublisher::lookup_datawriter_by_name (p. 1839)

DDSSubscriber::lookup_datareader_by_name (p. 1880)

9.254.2.35 create_participant_from_config_w_params()

```
virtual DDSDomainParticipant * DDSDomainParticipantFactory::create_participant_from_config_w_↩
params (
            const char * configuration_name,
            const DDS_DomainParticipantConfigParams_t & params ) [pure virtual]
```

<<***extension***>> *(p. 432)* Creates a **DDSDomainParticipant** (p. 1619) given its configuration name from a description provided in an XML configuration file and a set of parameters that allow changing some properties of such configuration.

The operation will create a **DDSDomainParticipant** (p. 1619) the same way specified in **DDSDomainParticipant↩ Factory::create_participant_from_config** (p. 1717).

In addition, this operation allows overriding the domain ID, participant name, and entities QoS specified in the configuration.

MT Safety:

UNSAFE. On VxWorks, when enabling the RTI Security Plugins, it is not safe for multiple threads to attempt to create a **DDSDomainParticipant** (p. 1619) when there are no other DomainParticipants in the **DDSDomain↩ ParticipantFactory** (p. 1694).

Parameters

configuration_name	<<***in***>> *(p. 433)* Name of the participant configuration in the XML file.
params	<<***in***>> *(p. 433)* input parameters that allow changing some properties of the configuration referred to by `configuration_name`.

Returns

The created **DDSDomainParticipant** (p. 1619) or NULL on error.

9.254.2.36 lookup_participant_by_name()

```
virtual DDSDomainParticipant * DDSDomainParticipantFactory::lookup_participant_by_name (
            const char * participant_name ) [pure virtual]
```

<<***extension***>> *(p. 432)* Looks up a **DDSDomainParticipant** (p. 1619) by its entity name in the **DDSDomain↩ ParticipantFactory** (p. 1694).

Every **DDSDomainParticipant** (p. 1619) in the system has an entity name which is configured and stored in the Entity↩ Name policy, **ENTITY_NAME** (p. 601).

This operation retrieves a **DDSDomainParticipant** (p. 1619) within the **DDSDomainParticipantFactory** (p. 1694) given the entity's name. If there are several **DDSDomainParticipant** (p. 1619) with the same name within the **DDSDomain↩ ParticipantFactory** (p. 1694) this function returns the first matching occurrence.

Parameters

participant_name	<<***in***>> *(p. 433)* Entity name of the **DDSDomainParticipant** (p. 1619). Cannot be NULL.

Returns

The first **DDSDomainParticipant** (p. 1619) found with the specified name or NULL if it is not found.

9.254.2.37 register_type_support()

```
virtual DDS_ReturnCode_t DDSDomainParticipantFactory::register_type_support (
            DDSDomainParticipantFactory_RegisterTypeFunction register_type_fcn,
            const char * type_name ) [pure virtual]
```

<<***extension***>> *(p. 432)* Registers a **DDSTypeSupport** (p. 1897) with the **DDSDomainParticipantFactory** (p. 1694) to enable automatic registration if the corresponding type, should it be needed by a **DDSDomainParticipant** (p. 1619).

Types refered by the **DDSTopic** (p. 1885) entities within a **DDSDomainParticipant** (p. 1619) must be registered with the **DDSDomainParticipant** (p. 1619).

Type registration in a **DDSDomainParticipant** (p. 1619) is performed by a call to the **FooTypeSupport::register_type** (p. 1981) operation. This can be done directly from the application code or indirectly by the RTI Connext infrastructure as a result of parsing an XML configuration file that refers to that type.

The **DDSDomainParticipantFactory::register_type_support** (p. 1720) operation provides the **DDSDomain↩ ParticipantFactory** (p. 1694) with the information it needs to automatically call the **FooTypeSupport::register_type** (p. 1981) operation and register the corresponding type if the type is needed as a result of parsing an XML configuration file.

Automatic type registration while parsing XML files can also be done by the RTI Connext infrastructure based on the type description provided in the XML files. If the **DDSTypeSupport** (p. 1897) has been registered with the **DDSDomain↩ ParticipantFactory** (p. 1694) this definition takes precedence over the description of the type given in the XML file.

The **DDSDomainParticipantFactory::register_type_support** (p. 1720) operation receives a **FooTypeSupport↩ ::register_type** (p. 1981) function as a parameter. This function is normally generated using rtiddsgen from a description of the corresponsing type in IDL, XML, or XSD.

The typical workflow when using this function is as follows: Define the data-type in IDL (or XML, or XSD) in a file. E.g. Foo.idl Run rtiddsgen in that file to generate the TypeSupport files, for the desired programming language. E.g. in C++ FooTypeSupport.h FooTypeSupport.cxx Include the proper header file (e.g. FooTypeSupport.h) in your program and call **DDSDomainParticipantFactory::register_type_support** (p. 1720) passing the function that was generated by rtiddsgen. E.g. **FooTypeSupport::register_type** (p. 1981) Include the TypeSupport source file in your project such that it is compiled and linked. E.g. FooTypeSupport.cxx

You may refer to the `Getting Started Guide` for additional details in this process.

Note that only one register function is allowed per registered type name.

Parameters

register_type_fcn	<<***in***>> *(*p. *433)* **DDSDomainParticipantFactory_RegisterTypeFunction** (p. 44) to be used for registering the type with a **DDSDomainParticipant** (p. 1619).
type_name	<<***in***>> *(*p. *433)* Name the type is registered with.

Exceptions

One	of the **Standard Return Codes** (p. 532)

9.254.2.38 get_participants()

```
virtual  DDS_ReturnCode_t DDSDomainParticipantFactory::get_participants (
             DDSDomainParticipantSeq & participants )  [pure virtual]
```

<<***extension***>> *(*p. *432)* Allows the application to access all the participants the DomainParticipantFactory has.

If the sequence doesn't own its buffer, and its maximum is less than the total number of participants, it will be filled up to its maximum, and fail with **DDS_RETCODE_OUT_OF_RESOURCES** (p. 533).

DomainParticipants are ordered in the output sequence from the most recently created to the oldest.

MT Safety:

Unsafe. Creating or deleting DomainParticipants at the same time as calling this operation is not safe.

Parameters

participants	<<***inout***>> *(*p. *433)* a DomainParticipantSeq object where the set or list of participants will be returned

Returns

One of the **Standard Return Codes** (p. 532) or **DDS_RETCODE_OUT_OF_RESOURCES** (p. 533)

9.254.2.39 set_thread_factory()

```
virtual  DDS_ReturnCode_t DDSDomainParticipantFactory::set_thread_factory (
              DDSThreadFactory * thread_factory )  [pure virtual]
```

*<<**extension**>> (p. 432)* Sets a **DDSThreadFactory** (p. 1883) in the **DDSDomainParticipantFactory** (p. 1694) that will be used to create the internal threads of the DDS middleware.

DDS threads are managed by the **DDSDomainParticipant** (p. 1619), which is responsible for creating and deleting threads throughout its lifecycle. Some threads are created when the **DDSDomainParticipant** (p. 1619) gets enabled (i.e., database, event, etc.) and others when special features are enabled (i.e., asynchronous publication thread).

DDS threads have **DDSDomainParticipant** (p. 1619) scope. This means every DomainParticipant creates all the necessary threads to operate and each **DDSDomainParticipant** (p. 1619) will create its own independent set of threads.

Each **DDSDomainParticipant** (p. 1619) creates an independent set of DDS threads and will use the **DDSThread↩ Factory** (p. 1883) that is held by the **DDSDomainParticipantFactory** (p. 1694) at the time the DomainParticipant is created. A **DDSThreadFactory** (p. 1883) is immutable from a **DDSDomainParticipant** (p. 1619) point of view. This means that a **DDSDomainParticipant** (p. 1619) will use throughout its lifecycle the same **DDSThreadFactory** (p. 1883) that was set when it was created, even if a new **DDSThreadFactory** (p. 1883) is set in the **DDSDomainParticipant↩ Factory** (p. 1694) later. That is, if a new **DDSThreadFactory** (p. 1883) is set, a previously created **DDSDomain↩ Participant** (p. 1619) will not be affected.

ThreadFactory lifecycle: A **DDSThreadFactory** (p. 1883) instance must be alive while there are DomainParticipants using it. A **DDSThreadFactory** (p. 1883) instance can be deleted only after all DomainParticipants using it are deleted.

By default, the **DDSDomainParticipantFactory** (p. 1694) has an internal **DDSThreadFactory** (p. 1883) so threads are created automatically by the core product. These threads are usually created using the typical framework for the target platform (i.e., pthread for Linux systems).

MT Safety:

: Safe. Creation and deletion of a **DDSDomainParticipant** (p. 1619) can occur concurrently with setting a **DDSThreadFactory** (p. 1883).

Parameters

thread_factory	*<<**in**>> (p. 433)* Instance of a **DDSThreadFactory** (p. 1883) to be used for creating the DDS threads. If null is specified, the current **DDSThreadFactory** (p. 1883) is removed and the internal default will be used.

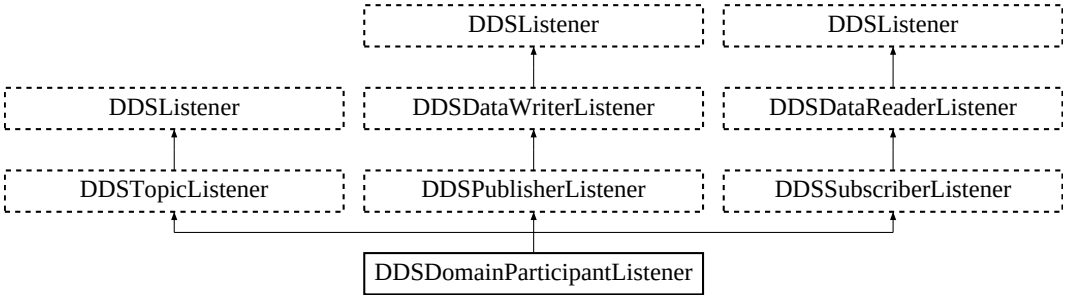
Exceptions

One	of the **Standard Return Codes** (p. 532)

9.255 DDSDomainParticipantListener Class Reference

*<<**interface**>> (p. 433)* Listener for participant status.

Inheritance diagram for DDSDomainParticipantListener:



Public Member Functions

- virtual void **on_invalid_local_status_advance_notice** (**DDSDomainParticipant** *participant, const **DDS_↩ InvalidLocalAdvanceNoticeStatus** &status)

  *Notifies the user that one of the local **DDSDomainParticipant** (p. 1619) trusted artifacts (identity or permissions) is about to expire.*

9.255.1 Detailed Description

<<***interface***>> *(p. 433)* Listener for participant status.

Entity:

DDSDomainParticipant (p. 1619)

Status:

Status Kinds (p. 533)

This is the interface that can be implemented by an application-provided class and then registered with the **DDSDomainParticipant** (p. 1619) such that the application can be notified by RTI Connext of relevant status changes.

The **DDSDomainParticipantListener** (p. 1722) interface extends all other Listener interfaces and has no additional operation beyond the ones defined by the more general listeners.

The purpose of the **DDSDomainParticipantListener** (p. 1722) is to be the listener of last resort that is notified of all status changes not captured by more specific listeners attached to the **DDSDomainEntity** (p. 1618) objects. When a relevant status change occurs, RTI Connext will first attempt to notify the listener attached to the concerned **DDSDomain↩ Entity** (p. 1618) if one is installed. Otherwise, RTI Connext will notify the Listener attached to the **DDSDomain↩ Participant** (p. 1619).

Important: Because a **DDSDomainParticipantListener** (p. 1722) may receive callbacks pertaining to many different entities, it is possible for the same listener to receive multiple callbacks simultaneously in different threads. (Such is not the case for listeners of other types.) It is therefore critical that users of this listener provide their own protection for any thread-unsafe activities undertaken in a **DDSDomainParticipantListener** (p. 1722) callback.

Note: Due to a thread-safety issue, the destruction of a DomainParticipantListener from an enabled DomainParticipant should be avoided – even if the DomainParticipantListener has been removed from the DomainParticipant. (This limitation does not affect the Java API.)

See also

DDSListener (p. 1794)

DDSDomainParticipant::set_listener (p. 1682)

9.255.2 Member Function Documentation

9.255.2.1 on_invalid_local_status_advance_notice()

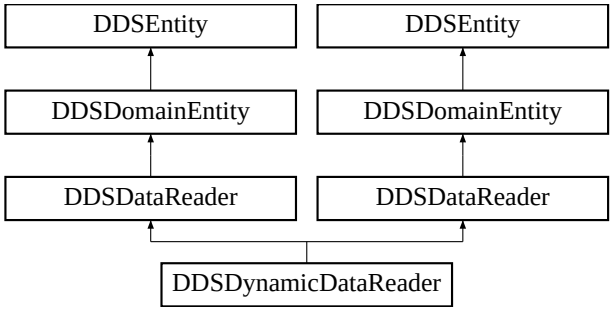
```
virtual void DDSDomainParticipantListener::on_invalid_local_status_advance_notice (
              DDSDomainParticipant * participant,
             const  DDS_InvalidLocalAdvanceNoticeStatus & status )  [virtual]
```

Notifies the user that one of the local **DDSDomainParticipant** (p. 1619) trusted artifacts (identity or permissions) is about to expire.

9.256 DDSDynamicDataReader Class Reference

Reads (subscribes to) objects of type **DDS_DynamicData** (p. 1013).

Inheritance diagram for DDSDynamicDataReader:



Additional Inherited Members

9.256.1 Detailed Description

Reads (subscribes to) objects of type **DDS_DynamicData** (p. 1013).

Instantiates **DDSDataReader** (p. 1556) < **DDS_DynamicData** (p. 1013) > .

See also

DDSDataReader (p. 1556)

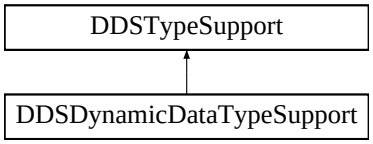
FooDataReader (p. 1917)

DDS_DynamicData (p. 1013)

9.257 DDSDynamicDataTypeSupport Class Reference

A factory for registering a dynamically defined type and creating **DDS_DynamicData** (p. 1013) objects.

Inheritance diagram for DDSDynamicDataTypeSupport:



Public Member Functions

- **DDS_Boolean is_valid** ()

  Indicates whether the object was constructed properly.
- **DDS_ReturnCode_t register_type** (**DDSDomainParticipant** ∗participant, const char ∗type_name)

  *Associate the **DDS_TypeCode** (p. 1428) with the given **DDSDomainParticipant** (p. 1619) under the given logical name.*
- **DDS_ReturnCode_t unregister_type** (**DDSDomainParticipant** ∗participant, const char ∗type_name)

  *Remove the definition of this type from the **DDSDomainParticipant** (p. 1619).*
- const char ∗ **get_type_name** () const

  Get the default name of this type.
- const **DDS_TypeCode** ∗ **get_data_type** () const

  *Get the **DDS_TypeCode** (p. 1428) wrapped by this **DDSDynamicDataTypeSupport** (p. 1725).*
- **DDS_DynamicData** ∗ **create_data** ()

  *Create a new **DDS_DynamicData** (p. 1013) sample initialized with the **DDS_TypeCode** (p. 1428) and properties of this **DDSDynamicDataTypeSupport** (p. 1725).*
- **DDS_ReturnCode_t delete_data** (**DDS_DynamicData** ∗a_data)

  *Finalize and deallocate the **DDS_DynamicData** (p. 1013) sample.*
- void **print_data** (const **DDS_DynamicData** ∗a_data) const

  Print a string representation of the given sample to the given file.
- **DDS_ReturnCode_t copy_data** (**DDS_DynamicData** ∗dest, const **DDS_DynamicData** ∗source) const

  Deeply copy the given data samples.
- **DDSDynamicDataTypeSupport** (const **DDS_TypeCode** ∗type, const struct **DDS_DynamicDataTypeProperty_t** &props)

  *Construct a new **DDSDynamicDataTypeSupport** (p. 1725) object.*
- virtual ∼**DDSDynamicDataTypeSupport** ()

  *Delete a **DDSDynamicDataTypeSupport** (p. 1725) object.*

9.257.1 Detailed Description

A factory for registering a dynamically defined type and creating **DDS_DynamicData** (p. 1013) objects.

A **DDSDynamicDataTypeSupport** (p. 1725) has three roles:

1. It associates a **DDS_TypeCode** (p. 1428) with policies for managing objects of that type. See the constructor, **DDSDynamicDataTypeSupport::DDSDynamicDataTypeSupport** (p. 1726).
2. It registers its type under logical names with a **DDSDomainParticipant** (p. 1619). See **DDSDynamicDataTypeSupport::register_type** (p. 1727).
3. It creates **DDS_DynamicData** (p. 1013) samples pre-initialized with the type and properties of the type support itself. See **DDSDynamicDataTypeSupport::create_data** (p. 1728).

9.257.2 Constructor & Destructor Documentation

9.257.2.1 DDSDynamicDataTypeSupport()

```
DDSDynamicDataTypeSupport::DDSDynamicDataTypeSupport (
            const  DDS_TypeCode * type,
            const struct  DDS_DynamicDataTypeProperty_t & props )
```

Construct a new **DDSDynamicDataTypeSupport** (p. 1725) object.

This step is usually followed by type registration.

NOTE that RTI Connext does not explicitly generate any exceptions in this constructor, because C++ exception support is not consistent across all platforms on which RTI Connext runs. Therefore, to check whether construction succeeded, you must use the **DDSDynamicDataTypeSupport::is_valid** (p. 1727) method.

The **DDS_TypeCode** (p. 1428) object that is passed to this constructor is cloned and stored internally; no pointer is retained to the object passed in. It is therefore safe to delete the **DDS_TypeCode** (p. 1428) after this method returns.

Parameters

type	The **DDS_TypeCode** (p. 1428) that describes the members of this type. The new object will contain a *copy* of this **DDS_TypeCode** (p. 1428); you may delete the argument after this constructor returns.
props	Policies that describe how to manage the memory and other properties of the data samples created by this factory. In most cases, the default values will be appropriate; see **DDS_DYNAMIC_DATA_TYPE_PROPERTY_DEFAULT** (p. 106).

See also

DDSDynamicDataTypeSupport::is_valid (p. 1727)

DDSDynamicDataTypeSupport::register_type (p. 1727)

DDSDynamicDataTypeSupport::~DDSDynamicDataTypeSupport (p. 1726)

Exceptions

One	of the **Standard Return Codes** (p. 532)

9.257.2.2 ~DDSDynamicDataTypeSupport()

```
virtual DDSDynamicDataTypeSupport::~DDSDynamicDataTypeSupport ( )  [virtual]
```

Delete a **DDSDynamicDataTypeSupport** (p. 1725) object.

A **DDSDynamicDataTypeSupport** (p. 1725) cannot be deleted while it is still in use. For each **DDSDomainParticipant** (p. 1619) with which the **DDSDynamicDataTypeSupport** (p. 1725) is registered, either the type must be unregistered or the participant must be deleted.

See also

DDSDynamicDataTypeSupport::unregister_type (p. 1727)

DDSDynamicDataTypeSupport::DDSDynamicDataTypeSupport (p. 1726)

9.257.3 Member Function Documentation

9.257.3.1 is_valid()

```
DDS_Boolean DDSDynamicDataTypeSupport::is_valid ( )
```

Indicates whether the object was constructed properly.

This method returns **DDS_BOOLEAN_TRUE** (p. 513) if the constructor succeeded; it returns **DDS_BOOLEAN_FALSE** (p. 513) if the constructor failed for any reason, which should also have resulted in a log message.

Possible failure reasons include passing an invalid type or invalid properties to the constructor.

This method is necessary because C++ exception support is not consistent across all of the platforms on which RTI Connext runs. Therefore, the implementation does not throw any exceptions in the constructor.

See also

DDSDynamicDataTypeSupport::DDSDynamicDataTypeSupport (p. 1726)

9.257.3.2 register_type()

```
DDS_ReturnCode_t DDSDynamicDataTypeSupport::register_type (
            DDSDomainParticipant * participant,
           const char * type_name )
```

Associate the **DDS_TypeCode** (p. 1428) with the given **DDSDomainParticipant** (p. 1619) under the given logical name.

Once a type has been registered, it can be referenced by name when creating a topic. Statically and dynamically defined types behave the same way in this respect.

If the type cannot be registered, this function will fail with **DDS_RETCODE_PRECONDITION_NOT_MET** (p. 532) or **DDS_RETCODE_OUT_OF_RESOURCES** (p. 533).

See also

FooTypeSupport::register_type (p. 1981)

DDSDomainParticipant::create_topic (p. 1650)

DDSDynamicDataTypeSupport::unregister_type (p. 1727)

9.257.3.3 unregister_type()

```
DDS_ReturnCode_t DDSDynamicDataTypeSupport::unregister_type (
            DDSDomainParticipant * participant,
            const char * type_name )
```

Remove the definition of this type from the **DDSDomainParticipant** (p. 1619).

This operation is optional; all types are automatically unregistered when a **DDSDomainParticipant** (p. 1619) is deleted. Most application will not need to manually unregister types.

A type cannot be unregistered while it is still in use; that is, while any **DDSTopic** (p. 1885) is still referring to it.

If the type cannot be unregistered, this function will fail with **DDS_RETCODE_ERROR** (p. 532) or **DDS_RETCODE_↩ BAD_PARAMETER** (p. 532).

See also

FooTypeSupport::unregister_type (p. 1982)

DDSDynamicDataTypeSupport::register_type (p. 1727)

9.257.3.4 get_type_name()

```
const char * DDSDynamicDataTypeSupport::get_type_name ( ) const
```

Get the default name of this type.

The **DDS_TypeCode** (p. 1428) that is wrapped by this **DDSDynamicDataTypeSupport** (p. 1725) includes a name; this operation returns that name.

This operation is useful when registering a type, because in most cases it is not necessary for the physical and logical names of the type to be different.

```
myTypeSupport->register_type(myParticipant, myTypeSupport->get_type_name());
```

See also

FooTypeSupport::get_type_name (p. 1988)

9.257.3.5 get_data_type()

```
const  DDS_TypeCode * DDSDynamicDataTypeSupport::get_data_type ( ) const
```

Get the **DDS_TypeCode** (p. 1428) wrapped by this **DDSDynamicDataTypeSupport** (p. 1725).

9.257.3.6 create_data()

```
DDS_DynamicData * DDSDynamicDataTypeSupport::create_data ( )
```

Create a new **DDS_DynamicData** (p. 1013) sample initialized with the **DDS_TypeCode** (p. 1428) and properties of this **DDSDynamicDataTypeSupport** (p. 1725).

You must delete your **DDS_DynamicData** (p. 1013) object when you are finished with it.

```
DDS_DynamicData* sample = myTypeSupport->create_data();

// Do something...

myTypeSupport->delete_data(sample);
```

See also

DDSDynamicDataTypeSupport::delete_data (p. 1729)

FooTypeSupport::create_data (p. 1983)

DDS_DynamicData::DDS_DynamicData(const DDS_TypeCode *, const DDS_DynamicDataProperty_t &) (p. 1029)

DDS_DynamicDataTypeProperty_t::data (p. 1139)

9.257.3.7 delete_data()

```
DDS_ReturnCode_t DDSDynamicDataTypeSupport::delete_data (
            DDS_DynamicData * a_data )
```

Finalize and deallocate the **DDS_DynamicData** (p. 1013) sample.

See also

FooTypeSupport::delete_data (p. 1984)

DDSDynamicDataTypeSupport::create_data (p. 1728)

9.257.3.8 print_data()

```
void DDSDynamicDataTypeSupport::print_data (
            const  DDS_DynamicData * a_data ) const
```

Print a string representation of the given sample to the given file.

This method is equivalent to **DDS_DynamicData::print** (p. 1040).

See also

DDS_DynamicData::print (p. 1040)

9.257.3.9 copy_data()

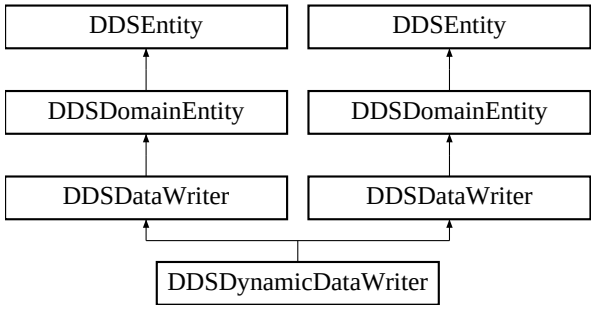
```
DDS_ReturnCode_t DDSDynamicDataTypeSupport::copy_data (
            DDS_DynamicData * dest,
          const  DDS_DynamicData * source ) const
```

Deeply copy the given data samples.

9.258 DDSDynamicDataWriter Class Reference

Writes (publishes) objects of type **DDS_DynamicData** (p. 1013).

Inheritance diagram for DDSDynamicDataWriter:



Additional Inherited Members

9.258.1 Detailed Description

Writes (publishes) objects of type **DDS_DynamicData** (p. 1013).

Instantiates **DDSDataWriter** (p. 1589) < **DDS_DynamicData** (p. 1013) > .

See also

DDSDataWriter (p. 1589)

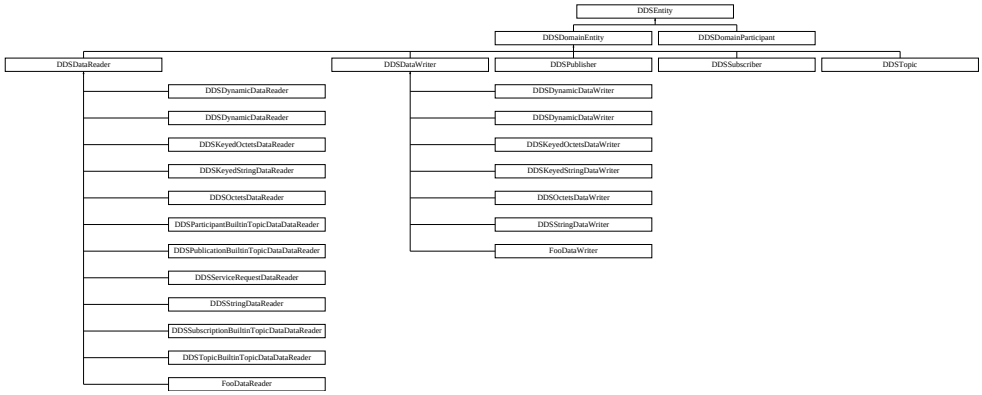
FooDataWriter (p. 1944)

DDS_DynamicData (p. 1013)

9.259 DDSEntity Class Reference

<<***interface***>> *(*p. *433)* Abstract base class for all the DDS objects that support QoS policies, a listener, and a status condition.

Inheritance diagram for DDSEntity:



Public Member Functions

- virtual **DDS_ReturnCode_t enable** ()=0

  *Enables the **DDSEntity** (*p. *1731).*
- virtual **DDSStatusCondition** ∗ **get_statuscondition** ()=0

  *Allows access to the **DDSStatusCondition** (*p. *1846) associated with the **DDSEntity** (*p. *1731).*
- virtual **DDS_StatusMask get_status_changes** ()=0

  *Retrieves the list of communication statuses in the **DDSEntity** (*p. *1731) that are triggered.*
- virtual **DDS_InstanceHandle_t get_instance_handle** ()=0

  *Allows access to the **DDS_InstanceHandle_t** (*p. *76) associated with the **DDSEntity** (*p. *1731).*

9.259.1 Detailed Description

<<***interface***>> *(*p. *433)* Abstract base class for all the DDS objects that support QoS policies, a listener, and a status condition.

All operations except for set_qos(), get_qos(), set_listener(), get_listener() and **enable()** (p. 1734), may return the value **DDS_RETCODE_NOT_ENABLED** (p. 533).

QoS:

All QoS Policies (p. 549)

Status:

Status Kinds (p. 533)

Listener:

DDSListener (p. 1794)

9.259.2 Abstract operations

Each derived entity provides the following operations specific to its role in RTI Connext.

9.259.2.1 set_qos (abstract)

This operation sets the QoS policies of the **DDSEntity** (p. 1731). Each of the derived entity classes provides this operation: **DDSEntity** (p. 1731) classes (**DDSDomainParticipant** (p. 1619), **DDSTopic** (p. 1885), **DDSPublisher** (p. 1818), **DDSDataWriter** (p. 1589), **DDSSubscriber** (p. 1860), and **DDSDataReader** (p. 1556)) so that the policies that are meaningful to each **DDSEntity** (p. 1731) can be set. For example, see **DDSDomainParticipant::set_qos** (p. 1677).

Precondition

Certain policies are immutable (see **All QoS Policies** (p. 549)): they can only be set at **DDSEntity** (p. 1731) creation time or before the entity is enabled. If `set_qos()` is invoked after the **DDSEntity** (p. 1731) is enabled and it attempts to change the value of an immutable policy, the operation will fail and return **DDS_RETCODE_IMMUTABLE_POLICY** (p. 533).

Certain values of QoS policies can be incompatible with the settings of the other policies. The `set_qos()` operation will also fail if it specifies a set of values that, once combined with the existing values, would result in an inconsistent set of policies. In this case, the operation will fail and return **DDS_RETCODE_INCONSISTENT_POLICY** (p. 533).

If the application supplies a non-default value for a QoS policy that is not supported by the implementation of the service, the set_qos operation will fail and return **DDS_RETCODE_UNSUPPORTED** (p. 532).

Postcondition

The existing set of policies is only changed if the `set_qos()` operation succeeds. This is indicated by a return code of **DDS_RETCODE_OK** (p. 532). In all other cases, none of the policies are modified.

Each derived **DDSEntity** (p. 1731) class (**DDSDomainParticipant** (p. 1619), **DDSTopic** (p. 1885), **DDSPublisher** (p. 1818), **DDSDataWriter** (p. 1589), **DDSSubscriber** (p. 1860), **DDSDataReader** (p. 1556)) has a corresponding special value of the QoS (**DDS_PARTICIPANT_QOS_DEFAULT** (p. 50), **DDS_PUBLISHER_QOS_DEFAULT** (p. 59), **DDS_SUBSCRIBER_QOS_DEFAULT** (p. 59), **DDS_TOPIC_QOS_DEFAULT** (p. 58), **DDS_DATAWRITER_QOS_DEFAULT** (p. 114), **DDS_DATAREADER_QOS_DEFAULT** (p. 136)). This special value may be used as a parameter to the `set_qos` operation to indicate that the QoS of the **DDSEntity** (p. 1731) should be changed to match the current default QoS set in the **DDSEntity** (p. 1731)'s factory. The operation set_qos cannot modify the immutable QoS, so a successful return of the operation indicates that the mutable QoS for the Entity has been modified to match the current default for the **DDSEntity** (p. 1731)'s factory.

The set of policies specified in the `qos` parameter are applied on top of the existing QoS, replacing the values of any policies previously set.

Possible error codes returned in addition to **Standard Return Codes** (p. 532) : **DDS_RETCODE_IMMUTABLE_POLICY** (p. 533), **DDS_RETCODE_INCONSISTENT_POLICY** (p. 533), **DDS_RETCODE_NOT_ALLOWED_BY_SECURITY** (p. 533).

9.259.2.2 get_qos (abstract)

This operation allows access to the existing set of QoS policies for the **DDSEntity** (p. 1731). This operation must be provided by each of the derived **DDSEntity** (p. 1731) classes (**DDSDomainParticipant** (p. 1619), **DDSTopic** (p. 1885), **DDSPublisher** (p. 1818), **DDSDataWriter** (p. 1589), **DDSSubscriber** (p. 1860), and **DDSDataReader** (p. 1556)), so that the policies that are meaningful to each **DDSEntity** (p. 1731) can be retrieved. For example, see **DDSDomain↩ Participant::get_qos** (p. 1678)

Possible error codes are **Standard Return Codes** (p. 532).

9.259.2.3 set_listener (abstract)

This operation installs a **DDSListener** (p. 1794) on the **DDSEntity** (p. 1731). The listener will only be invoked on the changes of communication status indicated by the specified `mask`.

This operation must be provided by each of the derived **DDSEntity** (p. 1731) classes (**DDSDomainParticipant** (p. 1619), **DDSTopic** (p. 1885), **DDSPublisher** (p. 1818), **DDSDataWriter** (p. 1589), **DDSSubscriber** (p. 1860), and **DDSData↩ Reader** (p. 1556)), so that the listener is of the concrete type suitable to the particular **DDSEntity** (p. 1731).

There are two components involved when setting up listeners: the listener itself and the mask. Both of these can be NULL.

Listeners for some Entities derive from the Connext DDS Listeners for related Entities. This means that the derived Listener has all of the methods of its parent class. You can install Listeners at all levels of the object hierarchy. At the top is the DomainParticipantListener; only one can be installed in a DomainParticipant. Then every Subscriber and Publisher can have their own Listener. Finally, each Topic, DataReader and DataWriter can have their own listeners. All are optional.

Suppose, however, that an Entity does not install a Listener, or installs a Listener that does not have particular communication status selected in the bitmask. In this case, if/when that particular status changes for that Entity, the corresponding Listener for that Entity's parent is called. Status changes are "propagated" from child Entity to parent Entity until a Listener is found that is registered for that status. Connext DDS will give up and drop the status-change event only if no Listeners have been installed in the object hierarchy to be called back for the specific status.

The following table describes the effect of different combinations of Listeners and Status Bit Masks considering the hierarchical processing.

Table 9.683 Effect of Different Combinations of Listeners and Status Bit Masks

	No Bits Set in Mask	Some/All Bits Set in Mask
Listener is Specified	Connext DDS finds the next most relevant listener for the changed status	For the statuses that are enabled in the mask, the most relevant listener will be called. The 'statusChangedFlag' for the relevant status is reset
Listener is NULL	Connext DDS behaves as if the listener is not installed and finds the next most relevant listener for that status	Connext DDS behaves as if the listener is installed, but the callback is doing nothing. This is called a "nil" listener

Postcondition

Only one listener can be attached to each **DDSEntity** (p. 1731). If a listener was already set, the operation `set_listener()` will replace it with the new one. Consequently, if the value NULL is passed for the listener parameter to the set_listener operation, any existing listener will be removed.

9.259.2.4 get_listener (abstract)

This operation allows access to the existing **DDSListener** (p. 1794) attached to the **DDSEntity** (p. 1731).

This operation must be provided by each of the derived **DDSEntity** (p. 1731) classes (**DDSDomainParticipant** (p. 1619), **DDSTopic** (p. 1885), **DDSPublisher** (p. 1818), **DDSDataWriter** (p. 1589), **DDSSubscriber** (p. 1860), and **DDSDataReader** (p. 1556)) so that the listener is of the concrete type suitable to the particular **DDSEntity** (p. 1731).

If no listener is installed on the **DDSEntity** (p. 1731), this operation will return NULL.

9.259.3 Member Function Documentation

9.259.3.1 enable()

```
virtual DDS_ReturnCode_t DDSEntity::enable ( ) [pure virtual]
```

Enables the **DDSEntity** (p. 1731).

This operation enables the Entity. Entity objects can be created either enabled or disabled. This is controlled by the value of the **ENTITY_FACTORY** (p. 600) QoS policy on the corresponding factory for the **DDSEntity** (p. 1731).

By default, **ENTITY_FACTORY** (p. 600) is set so that it is not necessary to explicitly call **DDSEntity::enable** (p. 1734) on newly created entities.

The **DDSEntity::enable** (p. 1734) operation is idempotent. Calling enable on an already enabled Entity returns OK and has no effect.

If a **DDSEntity** (p. 1731) has not yet been enabled, the following kinds of operations may be invoked on it:

- set or get the QoS policies (including default QoS policies) and listener
- **DDSEntity::get_statuscondition** (p. 1735)
- 'factory' operations
- **DDSEntity::get_status_changes** (p. 1735) and other get status operations (although the status of a disabled entity never changes)
- 'lookup' operations

Other operations may explicitly state that they may be called on disabled entities; those that do not will return the error **DDS_RETCODE_NOT_ENABLED** (p. 533).

It is legal to delete an **DDSEntity** (p. 1731) that has not been enabled by calling the proper operation on its factory .

Entities created from a factory Entity that is disabled are created disabled, regardless of the setting of the **DDS_Entity↩ FactoryQosPolicy** (p. 1145).

Calling enable on an Entity whose factory Entity is not enabled will fail and return **DDS_RETCODE_PRECONDITION↩ _NOT_MET** (p. 532).

If **DDS_EntityFactoryQosPolicy::autoenable_created_entities** (p. 1147) is TRUE, the enable operation on a factory will automatically enable all entities created from that factory (for example, enabling a **DDSPublisher** (p. 1818) will enable all its contained **DDSDataWriter** (p. 1589) objects)

Listeners associated with an entity are not called until the entity is enabled.

Conditions associated with a disabled entity are "inactive," that is, they have a `trigger_value` == FALSE.

Exceptions

One	of the **Standard Return Codes** (p. 532), **Standard Return Codes** (p. 532) or **DDS_RETCODE_PRECONDITION_NOT_MET** (p. 532).

Implemented in **DDSDataWriter** (p. 1610), and **DDSDataReader** (p. 1580).

9.259.3.2 get_statuscondition()

```
virtual DDSStatusCondition * DDSEntity::get_statuscondition ( ) [pure virtual]
```

Allows access to the **DDSStatusCondition** (p. 1846) associated with the **DDSEntity** (p. 1731).

The returned condition can then be added to a **DDSWaitSet** (p. 1897) so that the application can wait for specific status changes that affect the **DDSEntity** (p. 1731).

Returns

the status condition associated with this entity.

Implemented in **DDSDataWriter** (p. 1611), and **DDSDataReader** (p. 1581).

9.259.3.3 get_status_changes()

```
virtual DDS_StatusMask DDSEntity::get_status_changes ( ) [pure virtual]
```

Retrieves the list of communication statuses in the **DDSEntity** (p. 1731) that are triggered.

That is, the list of statuses whose value has changed since the last time the application read the status using the get_∗_status() method.

When the entity is first created or if the entity is not enabled, all communication statuses are in the "untriggered" state so the list returned by the get_status_changes operation will be empty.

The list of statuses returned by the get_status_changes operation refers to the status that are triggered on the Entity itself and does not include statuses that apply to contained entities.

Returns

list of communication statuses in the **DDSEntity** (p. 1731) that are triggered.

See also

Status Kinds (p. 533)

Implemented in **DDSDataWriter** (p. 1611), and **DDSDataReader** (p. 1582).

9.259.3.4 get_instance_handle()

```
virtual DDS_InstanceHandle_t DDSEntity::get_instance_handle ( ) [pure virtual]
```

Allows access to the **DDS_InstanceHandle_t** (p. 76) associated with the **DDSEntity** (p. 1731).

This operation returns the **DDS_InstanceHandle_t** (p. 76) that represents the **DDSEntity** (p. 1731).

Returns

the instance handle associated with this entity.

Implemented in **DDSDataWriter** (p. 1611), and **DDSDataReader** (p. 1582).

9.260 DDSFlowController Class Reference

<<***interface***>> *(*p. *433)* A flow controller is the object responsible for shaping the network traffic by determining when attached asynchronous **DDSDataWriter** (p. 1589) instances are allowed to write data.

Public Member Functions

- virtual **DDS_ReturnCode_t set_property** (const struct **DDS_FlowControllerProperty_t** &prop)=0

  *Sets the **DDSFlowController*** (p. *1736*) *property.*
- virtual **DDS_ReturnCode_t get_property** (struct **DDS_FlowControllerProperty_t** &prop)=0

  *Gets the **DDSFlowController*** (p. *1736*) *property.*
- virtual **DDS_ReturnCode_t trigger_flow** ()=0

  *Provides an external trigger to the **DDSFlowController*** (p. *1736*).
- virtual const char * **get_name** ()=0

  *Returns the name of the **DDSFlowController*** (p. *1736*).
- virtual **DDSDomainParticipant** * **get_participant** ()=0

  *Returns the **DDSDomainParticipant*** (p. *1619*) *to which the **DDSFlowController*** (p. *1736*) *belongs.*

9.260.1 Detailed Description

<<***interface***>> (p. *433*) A flow controller is the object responsible for shaping the network traffic by determining when attached asynchronous **DDSDataWriter** (p. 1589) instances are allowed to write data.

QoS:

DDS_FlowControllerProperty_t (p. 1156)

9.260.2 Member Function Documentation

9.260.2.1 set_property()

```
virtual DDS_ReturnCode_t DDSFlowController::set_property (
            const struct DDS_FlowControllerProperty_t & prop ) [pure virtual]
```

Sets the **DDSFlowController** (p. 1736) property.

This operation modifies the property of the **DDSFlowController** (p. 1736).

Once a **DDSFlowController** (p. 1736) has been instantiated, only the **DDS_FlowControllerProperty_t::token_bucket** (p. 1157) can be changed. The **DDS_FlowControllerProperty_t::scheduling_policy** (p. 1157) is immutable.

A new **DDS_FlowControllerTokenBucketProperty_t::period** (p. 1159) only takes effect at the next scheduled token distribution time (as determined by its previous value).

Parameters

prop	<<***in***>> *(*p. *433)* The new **DDS_FlowControllerProperty_t** (p. 1156). Property must be consistent. Immutable fields cannot be changed after **DDSFlowController** (p. 1736) has been created. The special value **DDS_FLOW_CONTROLLER_PROPERTY_DEFAULT** (p. 60) can be used to indicate that the property of the **DDSFlowController** (p. 1736) should be changed to match the current default **DDS_FlowControllerProperty_t** (p. 1156) set in the **DDSDomainParticipant** (p. 1619).

Exceptions

One	of the **Standard Return Codes** (p. 532), **DDS_RETCODE_IMMUTABLE_POLICY** (p. 533), or **DDS_RETCODE_INCONSISTENT_POLICY** (p. 533).

See also

DDS_FlowControllerProperty_t (p. 1156) for rules on consistency among property values.

9.260.2.2 get_property()

```
virtual  DDS_ReturnCode_t DDSFlowController::get_property (
             struct  DDS_FlowControllerProperty_t & prop )  [pure virtual]
```

Gets the **DDSFlowController** (p. 1736) property.

Parameters

prop	<<***in***>> *(*p. *433)* **DDSFlowController** (p. 1736) to be filled in.

Exceptions

One	of the **Standard Return Codes** (p. 532)

9.260.2.3 trigger_flow()

```
virtual  DDS_ReturnCode_t DDSFlowController::trigger_flow ( )  [pure virtual]
```

Provides an external trigger to the **DDSFlowController** (p. 1736).

Typically, a **DDSFlowController** (p. 1736) uses an internal trigger to periodically replenish its tokens. The period by which this trigger is called is determined by the **DDS_FlowControllerTokenBucketProperty_t::period** (p. 1159) property setting.

This function provides an additional, external trigger to the **DDSFlowController** (p. 1736). This trigger adds **DDS_↩ FlowControllerTokenBucketProperty_t::tokens_added_per_period** (p. 1158) tokens each time it is called (subject to the other property settings of the **DDSFlowController** (p. 1736)).

An *on-demand* **DDSFlowController** (p. 1736) can be created with a **DDS_DURATION_INFINITE** (p. 522) as **DDS_↩ FlowControllerTokenBucketProperty_t::period** (p. 1159), in which case the only trigger source is external (i.e. the FlowController is solely triggered by the user on demand).

DDSFlowController::trigger_flow (p. 1738) can be called on both strict *on-demand* FlowController and hybrid Flow↩ Controller (internally and externally triggered).

Exceptions

One	of the **Standard Return Codes** (p. 532)

9.260.2.4 get_name()

```
virtual const char * DDSFlowController::get_name ( )  [pure virtual]
```

Returns the name of the **DDSFlowController** (p. 1736).

Returns

The name of the **DDSFlowController** (p. 1736).

9.260.2.5 get_participant()

```
virtual  DDSDomainParticipant * DDSFlowController::get_participant ( )  [pure virtual]
```

Returns the **DDSDomainParticipant** (p. 1619) to which the **DDSFlowController** (p. 1736) belongs.

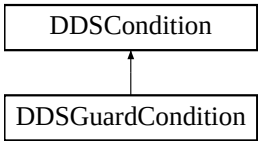
Returns

The **DDSDomainParticipant** (p. 1619) to which the **DDSFlowController** (p. 1736) belongs.

9.261 DDSGuardCondition Class Reference

<<***interface***>> *(p. 433)* A specific **DDSCondition** (p. 1544) whose `trigger_value` is completely under the control of the application.

Inheritance diagram for DDSGuardCondition:



Public Member Functions

- virtual **DDS_Boolean get_trigger_value** ()

  Retrieve the `trigger_value`*.*
- virtual **DDS_ReturnCode_t set_trigger_value** (**DDS_Boolean** value)

  Set the guard condition trigger value.
- **DDSGuardCondition** ()

  No argument constructor.
- virtual ~**DDSGuardCondition** ()

  Destructor.

9.261.1 Detailed Description

<<***interface***>> *(*p. *433)* A specific **DDSCondition** (p. 1544) whose `trigger_value` is completely under the control of the application.

The **DDSGuardCondition** (p. 1739) provides a way for an application to manually wake up a **DDSWaitSet** (p. 1897). This is accomplished by attaching the **DDSGuardCondition** (p. 1739) to the **DDSWaitSet** (p. 1897) and then setting the `trigger_value` by means of the **DDSGuardCondition::set_trigger_value** (p. 1741) operation.

See also

DDSWaitSet (p. 1897)

9.261.2 Constructor & Destructor Documentation

9.261.2.1 DDSGuardCondition()

```
DDSGuardCondition::DDSGuardCondition ( )
```

No argument constructor.

The default constructor initializes the guard condition with trigger value **DDS_BOOLEAN_FALSE** (p. 513)

9.261.2.2 ~DDSGuardCondition()

```
virtual DDSGuardCondition::~DDSGuardCondition ( )  [virtual]
```

Destructor.

Releases the resources associated with this object.

Deleting a NULL condition is safe and has no effect.

9.261.3 Member Function Documentation

9.261.3.1 get_trigger_value()

```
virtual  DDS_Boolean DDSGuardCondition::get_trigger_value ( )  [virtual]
```

Retrieve the `trigger_value`.

Returns

the trigger value.

Implements **DDSCondition** (p. 1545).

9.261.3.2 set_trigger_value()

```
virtual  DDS_ReturnCode_t DDSGuardCondition::set_trigger_value (
             DDS_Boolean value )  [virtual]
```

Set the guard condition trigger value.

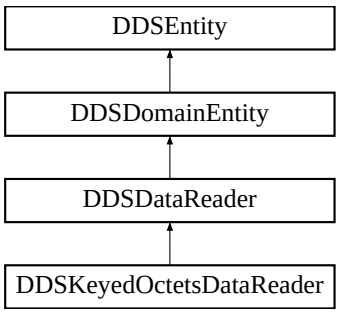
Parameters

value	<<***in***>> *(p. 433)* the new trigger value.

9.262 DDSKeyedOctetsDataReader Class Reference

<<***interface***>> *(p. 433)* Instantiates `DataReader` < **DDS_KeyedOctets** (p. 1168) >.

Inheritance diagram for DDSKeyedOctetsDataReader:



Public Member Functions

- virtual **DDS_ReturnCode_t read** (**DDS_KeyedOctetsSeq** &received_data, **DDS_SampleInfoSeq** &info_seq, **DDS_Long** max_samples= **DDS_LENGTH_UNLIMITED**, **DDS_SampleStateMask** sample_states= **DDS_ANY_SAMPLE_STATE**, **DDS_ViewStateMask** view_states= **DDS_ANY_VIEW_STATE**, **DDS_InstanceStateMask** instance_states= **DDS_ANY_INSTANCE_STATE**)

  *Access a collection of data samples from the **DDSDataReader** (p. 1556).*
- virtual **DDS_ReturnCode_t take** (**DDS_KeyedOctetsSeq** &received_data, **DDS_SampleInfoSeq** &info_seq, **DDS_Long** max_samples= **DDS_LENGTH_UNLIMITED**, **DDS_SampleStateMask** sample_states= **DDS_ANY_SAMPLE_STATE**, **DDS_ViewStateMask** view_states= **DDS_ANY_VIEW_STATE**, **DDS_InstanceStateMask** instance_states= **DDS_ANY_INSTANCE_STATE**)

  *Access a collection of data-samples from the **DDSDataReader** (p. 1556).*
- virtual **DDS_ReturnCode_t read_w_condition** (**DDS_KeyedOctetsSeq** &received_data, **DDS_SampleInfoSeq** &info_seq, **DDS_Long** max_samples, **DDSReadCondition** *condition)

  *Accesses via **DDSKeyedOctetsDataReader::read** (p. 1744) the samples that match the criteria specified in the **DDSReadCondition** (p. 1842).*
- virtual **DDS_ReturnCode_t take_w_condition** (**DDS_KeyedOctetsSeq** &received_data, **DDS_SampleInfoSeq** &info_seq, **DDS_Long** max_samples, **DDSReadCondition** *condition)

  *Analogous to **DDSKeyedOctetsDataReader::read_w_condition** (p. 1744) except it accesses samples via the **DDSKeyedOctetsDataReader::take** (p. 1744) operation.*
- virtual **DDS_ReturnCode_t read_next_sample** (**DDS_KeyedOctets** &received_data, **DDS_SampleInfo** &sample_info)

  *Copies the next not-previously-accessed data value from the **DDSDataReader** (p. 1556).*
- virtual **DDS_ReturnCode_t take_next_sample** (**DDS_KeyedOctets** &received_data, **DDS_SampleInfo** &sample_info)

  *Copies the next not-previously-accessed data value from the **DDSDataReader** (p. 1556).*
- virtual **DDS_ReturnCode_t read_instance** (**DDS_KeyedOctetsSeq** &received_data, **DDS_SampleInfoSeq** &info_seq, **DDS_Long** max_samples= **DDS_LENGTH_UNLIMITED**, const **DDS_InstanceHandle_t** &a_handle= **DDS_HANDLE_NIL**, **DDS_SampleStateMask** sample_states= **DDS_ANY_SAMPLE_STATE**, **DDS_ViewStateMask** view_states= **DDS_ANY_VIEW_STATE**, **DDS_InstanceStateMask** instance_states= **DDS_ANY_INSTANCE_STATE**)

  *Access a collection of data samples from the **DDSDataReader** (p. 1556).*
- virtual **DDS_ReturnCode_t take_instance** (**DDS_KeyedOctetsSeq** &received_data, **DDS_SampleInfoSeq** &info_seq, **DDS_Long** max_samples= **DDS_LENGTH_UNLIMITED**, const **DDS_InstanceHandle_t** &a_handle= **DDS_HANDLE_NIL**, **DDS_SampleStateMask** sample_states= **DDS_ANY_SAMPLE_STATE**, **DDS_ViewStateMask** view_states= **DDS_ANY_VIEW_STATE**, **DDS_InstanceStateMask** instance_states= **DDS_ANY_INSTANCE_STATE**)

  *Access a collection of data samples from the **DDSDataReader** (p. 1556).*
- virtual **DDS_ReturnCode_t read_instance_w_condition** (**DDS_KeyedOctetsSeq** &received_data, **DDS_SampleInfoSeq** &info_seq, **DDS_Long** max_samples= **DDS_LENGTH_UNLIMITED**, const **DDS_InstanceHandle_t** &a_handle= **DDS_HANDLE_NIL**, **DDSReadCondition** *condition=NULL)

  *Accesses via **DDSKeyedOctetsDataReader::read_instance** (p. 1745) the samples that match the criteria specified in the **DDSReadCondition** (p. 1842).*
- virtual **DDS_ReturnCode_t take_instance_w_condition** (**DDS_KeyedOctetsSeq** &received_data, **DDS_SampleInfoSeq** &info_seq, **DDS_Long** max_samples= **DDS_LENGTH_UNLIMITED**, const **DDS_InstanceHandle_t** &a_handle= **DDS_HANDLE_NIL**, **DDSReadCondition** *condition=NULL)

  *Accesses via **DDSKeyedOctetsDataReader::take_instance** (p. 1746) the samples that match the criteria specified in the **DDSReadCondition** (p. 1842).*
- virtual **DDS_ReturnCode_t read_next_instance** (**DDS_KeyedOctetsSeq** &received_data, **DDS_SampleInfoSeq** &info_seq, **DDS_Long** max_samples= **DDS_LENGTH_UNLIMITED**, const **DDS_InstanceHandle_t** &previous_handle= **DDS_HANDLE_NIL**, **DDS_SampleStateMask** sample_states= **DDS_ANY_SAMPLE_STATE**, **DDS_ViewStateMask** view_states= **DDS_ANY_VIEW_STATE**, **DDS_InstanceStateMask** instance_states= **DDS_ANY_INSTANCE_STATE**)

*Access a collection of data samples from the **DDSDataReader** (p. 1556).*

- virtual **DDS_ReturnCode_t take_next_instance** (**DDS_KeyedOctetsSeq** &received_data, **DDS_SampleInfoSeq** &info_seq, **DDS_Long** max_samples= **DDS_LENGTH_UNLIMITED**, const **DDS_InstanceHandle_t** &previous_handle= **DDS_HANDLE_NIL**, **DDS_SampleStateMask** sample_states= **DDS_ANY_SAMPLE_STATE**, **DDS_ViewStateMask** view_states= **DDS_ANY_VIEW_STATE**, **DDS_InstanceStateMask** instance_states= **DDS_ANY_INSTANCE_STATE**)

  *Access a collection of data samples from the **DDSDataReader** (p. 1556).*
- virtual **DDS_ReturnCode_t read_next_instance_w_condition** (**DDS_KeyedOctetsSeq** &received_data, **DDS_SampleInfoSeq** &info_seq, **DDS_Long** max_samples= **DDS_LENGTH_UNLIMITED**, const **DDS_InstanceHandle_t** &previous_handle= **DDS_HANDLE_NIL**, **DDSReadCondition** *condition=NULL)

  *Accesses via **DDSKeyedOctetsDataReader::read_next_instance** (p. 1747) the samples that match the criteria specified in the **DDSReadCondition** (p. 1842).*
- virtual **DDS_ReturnCode_t take_next_instance_w_condition** (**DDS_KeyedOctetsSeq** &received_data, **DDS_SampleInfoSeq** &info_seq, **DDS_Long** max_samples= **DDS_LENGTH_UNLIMITED**, const **DDS_InstanceHandle_t** &previous_handle= **DDS_HANDLE_NIL**, **DDSReadCondition** *condition=NULL)

  *Accesses via **DDSKeyedOctetsDataReader::take_next_instance** (p. 1747) the samples that match the criteria specified in the **DDSReadCondition** (p. 1842).*
- virtual **DDS_ReturnCode_t return_loan** (**DDS_KeyedOctetsSeq** &received_data, **DDS_SampleInfoSeq** &info_seq)

  *Indicates to the **DDSDataReader** (p. 1556) that the application is done accessing the collection of `received_data` and `info_seq` obtained by some earlier invocation of read or take on the **DDSDataReader** (p. 1556).*
- virtual **DDS_ReturnCode_t get_key_value** (**DDS_KeyedOctets** &key_holder, const **DDS_InstanceHandle_t** &handle)

  Retrieve the instance `key` that corresponds to an instance `handle`.
- virtual **DDS_ReturnCode_t get_key_value** (char *key, const **DDS_InstanceHandle_t** &handle)

  *<<**extension**>> (p. 432) Retrieve the instance `key` that corresponds to an instance `handle`.*
- virtual **DDS_InstanceHandle_t lookup_instance** (const **DDS_KeyedOctets** &key_holder)

  Retrieve the instance `handle` that corresponds to an instance `key_holder`.
- virtual **DDS_InstanceHandle_t lookup_instance** (const char *key)

  *<<**extension**>> (p. 432) Retrieve the instance `handle` that corresponds to an instance `key`.*

Static Public Member Functions

- static **DDSKeyedOctetsDataReader** * **narrow** (**DDSDataReader** *reader)

  *Narrow the given **DDSDataReader** (p. 1556) pointer to a **DDSKeyedOctetsDataReader** (p. 1741) pointer.*

9.262.1 Detailed Description

<<***interface***>> *(p. 433)* Instantiates `DataReader` < **DDS_KeyedOctets** (p. 1168) >.

When reading or taking data with this reader, if you request a copy of the samples instead of a loan, and a string or byte array in a destination data sample is NULL, the middleware will allocate a new string or array for you of sufficient length to hold the received data. New strings will be allocated with **DDS_String_alloc** (p. 759); new arrays will be allocated with **DDS_OctetBuffer_alloc** (p. 755). The sample's destructor will delete them.

A non- NULL string or array is assumed to be allocated to sufficient length to store the incoming data. It will not be reallocated.

See also

FooDataReader (p. 1917)

DDSDataReader (p. 1556)

9.262.2 Member Function Documentation

9.262.2.1 read()

```
virtual DDS_ReturnCode_t DDSKeyedOctetsDataReader::read (
            DDS_KeyedOctetsSeq & received_data,
            DDS_SampleInfoSeq & info_seq,
            DDS_Long max_samples = DDS_LENGTH_UNLIMITED,
            DDS_SampleStateMask sample_states = DDS_ANY_SAMPLE_STATE,
            DDS_ViewStateMask view_states = DDS_ANY_VIEW_STATE,
            DDS_InstanceStateMask instance_states = DDS_ANY_INSTANCE_STATE ) [virtual]
```

Access a collection of data samples from the **DDSDataReader** (p. 1556).

See also

FooDataReader::read (p. 1919)

9.262.2.2 take()

```
virtual DDS_ReturnCode_t DDSKeyedOctetsDataReader::take (
            DDS_KeyedOctetsSeq & received_data,
            DDS_SampleInfoSeq & info_seq,
            DDS_Long max_samples = DDS_LENGTH_UNLIMITED,
            DDS_SampleStateMask sample_states = DDS_ANY_SAMPLE_STATE,
            DDS_ViewStateMask view_states = DDS_ANY_VIEW_STATE,
            DDS_InstanceStateMask instance_states = DDS_ANY_INSTANCE_STATE ) [virtual]
```

Access a collection of data-samples from the **DDSDataReader** (p. 1556).

See also

FooDataReader::take (p. 1920)

9.262.2.3 read_w_condition()

```
virtual DDS_ReturnCode_t DDSKeyedOctetsDataReader::read_w_condition (
            DDS_KeyedOctetsSeq & received_data,
            DDS_SampleInfoSeq & info_seq,
            DDS_Long max_samples,
            DDSReadCondition * condition ) [virtual]
```

Accesses via **DDSKeyedOctetsDataReader::read** (p. 1744) the samples that match the criteria specified in the **DDSReadCondition** (p. 1842).

See also

FooDataReader::read_w_condition (p. 1925)

9.262.2.4 take_w_condition()

```
virtual DDS_ReturnCode_t DDSKeyedOctetsDataReader::take_w_condition (
            DDS_KeyedOctetsSeq & received_data,
            DDS_SampleInfoSeq & info_seq,
            DDS_Long max_samples,
            DDSReadCondition * condition ) [virtual]
```

Analogous to **DDSKeyedOctetsDataReader::read_w_condition** (p. 1744) except it accesses samples via the **DDSKeyedOctetsDataReader::take** (p. 1744) operation.

See also

FooDataReader::take_w_condition (p. 1926)

9.262.2.5 read_next_sample()

```
virtual DDS_ReturnCode_t DDSKeyedOctetsDataReader::read_next_sample (
            DDS_KeyedOctets & received_data,
            DDS_SampleInfo & sample_info ) [virtual]
```

Copies the next not-previously-accessed data value from the **DDSDataReader** (p. 1556).

See also

FooDataReader::read_next_sample (p. 1927)

9.262.2.6 take_next_sample()

```
virtual DDS_ReturnCode_t DDSKeyedOctetsDataReader::take_next_sample (
            DDS_KeyedOctets & received_data,
            DDS_SampleInfo & sample_info ) [virtual]
```

Copies the next not-previously-accessed data value from the **DDSDataReader** (p. 1556).

See also

FooDataReader::take_next_sample (p. 1928)

9.262.2.7 read_instance()

```
virtual  DDS_ReturnCode_t DDSKeyedOctetsDataReader::read_instance (
             DDS_KeyedOctetsSeq & received_data,
             DDS_SampleInfoSeq & info_seq,
             DDS_Long max_samples =  DDS_LENGTH_UNLIMITED,
            const  DDS_InstanceHandle_t & a_handle =  DDS_HANDLE_NIL,
             DDS_SampleStateMask sample_states =  DDS_ANY_SAMPLE_STATE,
             DDS_ViewStateMask view_states =  DDS_ANY_VIEW_STATE,
             DDS_InstanceStateMask instance_states =  DDS_ANY_INSTANCE_STATE )  [virtual]
```

Access a collection of data samples from the **DDSDataReader** (p. 1556).

See also

FooDataReader::read_instance (p. 1929)

9.262.2.8 take_instance()

```
virtual  DDS_ReturnCode_t DDSKeyedOctetsDataReader::take_instance (
             DDS_KeyedOctetsSeq & received_data,
             DDS_SampleInfoSeq & info_seq,
             DDS_Long max_samples =  DDS_LENGTH_UNLIMITED,
            const  DDS_InstanceHandle_t & a_handle =  DDS_HANDLE_NIL,
             DDS_SampleStateMask sample_states =  DDS_ANY_SAMPLE_STATE,
             DDS_ViewStateMask view_states =  DDS_ANY_VIEW_STATE,
             DDS_InstanceStateMask instance_states =  DDS_ANY_INSTANCE_STATE )  [virtual]
```

Access a collection of data samples from the **DDSDataReader** (p. 1556).

See also

FooDataReader::take_instance (p. 1931)

9.262.2.9 read_instance_w_condition()

```
virtual  DDS_ReturnCode_t DDSKeyedOctetsDataReader::read_instance_w_condition (
             DDS_KeyedOctetsSeq & received_data,
             DDS_SampleInfoSeq & info_seq,
             DDS_Long max_samples =  DDS_LENGTH_UNLIMITED,
            const  DDS_InstanceHandle_t & a_handle =  DDS_HANDLE_NIL,
             DDSReadCondition * condition = NULL )  [virtual]
```

Accesses via **DDSKeyedOctetsDataReader::read_instance** (p. 1745) the samples that match the criteria specified in the **DDSReadCondition** (p. 1842).

See also

FooDataReader::read_instance_w_condition (p. 1932)

9.262.2.10 take_instance_w_condition()

```
virtual  DDS_ReturnCode_t DDSKeyedOctetsDataReader::take_instance_w_condition (
              DDS_KeyedOctetsSeq & received_data,
              DDS_SampleInfoSeq & info_seq,
              DDS_Long max_samples =  DDS_LENGTH_UNLIMITED,
             const  DDS_InstanceHandle_t & a_handle =  DDS_HANDLE_NIL,
              DDSReadCondition * condition = NULL )  [virtual]
```

Accesses via **DDSKeyedOctetsDataReader::take_instance** (p. 1746) the samples that match the criteria specified in the **DDSReadCondition** (p. 1842).

See also

FooDataReader::take_instance_w_condition (p. 1933)

9.262.2.11 read_next_instance()

```
virtual  DDS_ReturnCode_t DDSKeyedOctetsDataReader::read_next_instance (
              DDS_KeyedOctetsSeq & received_data,
              DDS_SampleInfoSeq & info_seq,
              DDS_Long max_samples =  DDS_LENGTH_UNLIMITED,
             const  DDS_InstanceHandle_t & previous_handle =  DDS_HANDLE_NIL,
              DDS_SampleStateMask sample_states =  DDS_ANY_SAMPLE_STATE,
              DDS_ViewStateMask view_states =  DDS_ANY_VIEW_STATE,
              DDS_InstanceStateMask instance_states =  DDS_ANY_INSTANCE_STATE )  [virtual]
```

Access a collection of data samples from the **DDSDataReader** (p. 1556).

See also

FooDataReader::read_next_instance (p. 1935)

9.262.2.12 take_next_instance()

```
virtual  DDS_ReturnCode_t DDSKeyedOctetsDataReader::take_next_instance (
              DDS_KeyedOctetsSeq & received_data,
              DDS_SampleInfoSeq & info_seq,
              DDS_Long max_samples =  DDS_LENGTH_UNLIMITED,
             const  DDS_InstanceHandle_t & previous_handle =  DDS_HANDLE_NIL,
              DDS_SampleStateMask sample_states =  DDS_ANY_SAMPLE_STATE,
              DDS_ViewStateMask view_states =  DDS_ANY_VIEW_STATE,
              DDS_InstanceStateMask instance_states =  DDS_ANY_INSTANCE_STATE )  [virtual]
```

Access a collection of data samples from the **DDSDataReader** (p. 1556).

See also

FooDataReader::take_next_instance (p. 1936)

9.262.2.13 read_next_instance_w_condition()

```
virtual DDS_ReturnCode_t DDSKeyedOctetsDataReader::read_next_instance_w_condition (
            DDS_KeyedOctetsSeq & received_data,
            DDS_SampleInfoSeq & info_seq,
            DDS_Long max_samples = DDS_LENGTH_UNLIMITED,
           const DDS_InstanceHandle_t & previous_handle = DDS_HANDLE_NIL,
            DDSReadCondition * condition = NULL ) [virtual]
```

Accesses via **DDSKeyedOctetsDataReader::read_next_instance** (p. 1747) the samples that match the criteria specified in the **DDSReadCondition** (p. 1842).

See also

FooDataReader::read_next_instance_w_condition (p. 1938)

9.262.2.14 take_next_instance_w_condition()

```
virtual DDS_ReturnCode_t DDSKeyedOctetsDataReader::take_next_instance_w_condition (
            DDS_KeyedOctetsSeq & received_data,
            DDS_SampleInfoSeq & info_seq,
            DDS_Long max_samples = DDS_LENGTH_UNLIMITED,
           const DDS_InstanceHandle_t & previous_handle = DDS_HANDLE_NIL,
            DDSReadCondition * condition = NULL ) [virtual]
```

Accesses via **DDSKeyedOctetsDataReader::take_next_instance** (p. 1747) the samples that match the criteria specified in the **DDSReadCondition** (p. 1842).

See also

FooDataReader::take_next_instance_w_condition (p. 1939)

9.262.2.15 return_loan()

```
virtual DDS_ReturnCode_t DDSKeyedOctetsDataReader::return_loan (
            DDS_KeyedOctetsSeq & received_data,
            DDS_SampleInfoSeq & info_seq ) [virtual]
```

Indicates to the **DDSDataReader** (p. 1556) that the application is done accessing the collection of `received_data` and `info_seq` obtained by some earlier invocation of read or take on the **DDSDataReader** (p. 1556).

See also

FooDataReader::return_loan (p. 1940)

9.262.2.16 get_key_value() [1/2]

```
virtual  DDS_ReturnCode_t DDSKeyedOctetsDataReader::get_key_value (
             DDS_KeyedOctets & key_holder,
             const  DDS_InstanceHandle_t & handle )  [virtual]
```

Retrieve the instance `key` that corresponds to an instance `handle`.

See also

FooDataReader::get_key_value (p. 1941)

9.262.2.17 get_key_value() [2/2]

```
virtual  DDS_ReturnCode_t DDSKeyedOctetsDataReader::get_key_value (
             char * key,
             const  DDS_InstanceHandle_t & handle )  [virtual]
```

<<***extension***>> *(p. 432)* Retrieve the instance `key` that corresponds to an instance `handle`.

See also

FooDataReader::get_key_value (p. 1941)

9.262.2.18 lookup_instance() [1/2]

```
virtual  DDS_InstanceHandle_t DDSKeyedOctetsDataReader::lookup_instance (
             const  DDS_KeyedOctets & key_holder )  [virtual]
```

Retrieve the instance `handle` that corresponds to an instance `key_holder`.

See also

FooDataReader::lookup_instance (p. 1942)

9.262.2.19 lookup_instance() [2/2]

```
virtual DDS_InstanceHandle_t DDSKeyedOctetsDataReader::lookup_instance (
            const char * key ) [virtual]
```

<<***extension***>> *(p. 432)* Retrieve the instance `handle` that corresponds to an instance `key`.

See also

FooDataReader::lookup_instance (p. 1942)

9.262.2.20 narrow()

```
static DDSKeyedOctetsDataReader * DDSKeyedOctetsDataReader::narrow (
             DDSDataReader * reader ) [static]
```

Narrow the given **DDSDataReader** (p. 1556) pointer to a **DDSKeyedOctetsDataReader** (p. 1741) pointer.

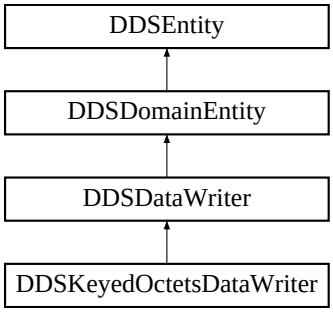
See also

FooDataReader::narrow (p. 1919)

9.263 DDSKeyedOctetsDataWriter Class Reference

<<***interface***>> *(p. 433)* Instantiates `DataWriter` < **DDS_KeyedOctets** (p. 1168) >.

Inheritance diagram for DDSKeyedOctetsDataWriter:



Public Member Functions

- virtual **DDS_InstanceHandle_t register_instance** (const **DDS_KeyedOctets** &instance_data)

  Informs RTI Connext that the application will be modifying a particular instance.
- virtual **DDS_InstanceHandle_t register_instance** (const char *key)

  <<***extension***>> (p. *432*) *Informs RTI Connext that the application will be modifying a particular instance.*
- virtual **DDS_InstanceHandle_t register_instance_w_timestamp** (const **DDS_KeyedOctets** &instance_data, const **DDS_Time_t** &source_timestamp)

  *Performs the same functions as **DDSKeyedOctetsDataWriter::register_instance** (p. 1753) except that the application provides the value for the* `source_timestamp`.
- virtual **DDS_InstanceHandle_t register_instance_w_timestamp** (const char *key, const **DDS_Time_t** &source_timestamp)

  <<***extension***>> (p. *432*) *Performs the same functions as **DDSKeyedOctetsDataWriter::register_instance** (p. 1753) except that the application provides the value for the* `source_timestamp`.
- virtual **DDS_ReturnCode_t unregister_instance** (const **DDS_KeyedOctets** &instance_data, const **DDS_InstanceHandle_t** &handle)

  *Reverses the action of **DDSKeyedOctetsDataWriter::register_instance** (p. 1753).*
- virtual **DDS_ReturnCode_t unregister_instance** (const char *key, const **DDS_InstanceHandle_t** &handle)

  <<***extension***>> (p. *432*) *Reverses the action of **DDSKeyedOctetsDataWriter::register_instance** (p. 1753).*
- virtual **DDS_ReturnCode_t unregister_instance_w_timestamp** (const **DDS_KeyedOctets** &instance_data, const **DDS_InstanceHandle_t** &handle, const **DDS_Time_t** &source_timestamp)

  *Performs the same function as **DDSKeyedOctetsDataWriter::unregister_instance** (p. 1754) except that it also provides the value for the* `source_timestamp`.
- virtual **DDS_ReturnCode_t unregister_instance_w_timestamp** (const char *key, const **DDS_InstanceHandle_t** &handle, const **DDS_Time_t** &source_timestamp)

  *Performs the same function as **DDSKeyedOctetsDataWriter::unregister_instance** (p. 1754) except that it also provides the value for the* `source_timestamp`.
- virtual **DDS_KeyedOctets** * **create_data** (const **DDS_TypeAllocationParams_t** &alloc_params)

  Creates a keyed octet sequence.
- virtual **DDS_Boolean delete_data** (**DDS_KeyedOctets** *sample, const **DDS_TypeDeallocationParams_t** &dealloc_params)

  *Destroys a string data instance created by **DDSStringDataWriter::create_data** (p. 1853).*
- virtual **DDS_ReturnCode_t write** (const **DDS_KeyedOctets** &instance_data, const **DDS_InstanceHandle_t** &handle)

  *Modifies the value of a **DDS_KeyedOctets** (p. 1168) data instance.*
- virtual **DDS_ReturnCode_t write** (const char *key, const unsigned char *octets, int length, const **DDS_InstanceHandle_t** &handle)

  <<***extension***>> (p. *432*) *Modifies the value of a **DDS_KeyedOctets** (p. 1168) data instance.*
- virtual **DDS_ReturnCode_t write** (const char *key, const **DDS_OctetSeq** &octets, const **DDS_InstanceHandle_t** &handle)

  <<***extension***>> (p. *432*) *Modifies the value of a **DDS_KeyedOctets** (p. 1168) data instance.*
- virtual **DDS_ReturnCode_t write_w_timestamp** (const **DDS_KeyedOctets** &instance_data, const **DDS_InstanceHandle_t** &handle, const **DDS_Time_t** &source_timestamp)

  *Performs the same function as **DDSKeyedOctetsDataWriter::write** (p. 1756) except that it also provides the value for the* `source_timestamp`.
- virtual **DDS_ReturnCode_t write_w_timestamp** (const char *key, const unsigned char *octets, int length, const **DDS_InstanceHandle_t** &handle, const **DDS_Time_t** &source_timestamp)

  *Performs the same function as **DDSKeyedOctetsDataWriter::write** (p. 1756) except that it also provides the value for the* `source_timestamp`.

- virtual **DDS_ReturnCode_t write_w_timestamp** (const char *key, const **DDS_OctetSeq** &octets, const **DDS_InstanceHandle_t** &handle, const **DDS_Time_t** &source_timestamp)

  *Performs the same function as **DDSKeyedOctetsDataWriter::write** (p. 1756) except that it also provides the value for the `source_timestamp`.*
- virtual **DDS_ReturnCode_t write_w_params** (const **DDS_KeyedOctets** &instance_data, **DDS_WriteParams_t** &params)

  *Performs the same function as **DDSKeyedOctetsDataWriter::write** (p. 1756) except that it also allows specification of the instance handle, source timestamp, publication priority, and cookie.*
- virtual **DDS_ReturnCode_t write_w_params** (const char *key, const unsigned char *octets, int length, **DDS_WriteParams_t** &params)

  *Performs the same function as **DDSKeyedOctetsDataWriter::write** (p. 1756) except that it also allows specification of the instance handle, source timestamp, publication priority, and cookie.*
- virtual **DDS_ReturnCode_t write_w_params** (const char *key, const **DDS_OctetSeq** &octets, **DDS_WriteParams_t** &params)

  *Performs the same function as **DDSKeyedOctetsDataWriter::write** (p. 1756) except that it also allows specification of the instance handle, source timestamp, publication priority, and cookie.*
- virtual **DDS_ReturnCode_t dispose** (const **DDS_KeyedOctets** &instance_data, const **DDS_InstanceHandle_t** &handle)

  Requests the middleware to delete the data.
- virtual **DDS_ReturnCode_t dispose** (const char *key, const **DDS_InstanceHandle_t** &instance_handle)

  *<<**extension**>> (p. 432) Requests the middleware to delete the data.*
- virtual **DDS_ReturnCode_t dispose_w_timestamp** (const **DDS_KeyedOctets** &instance_data, const **DDS_InstanceHandle_t** &handle, const **DDS_Time_t** &source_timestamp)

  *Performs the same functions as **DDSKeyedOctetsDataWriter::dispose** (p. 1761) except that the application provides the value for the `source_timestamp` that is made available to **DDSDataReader** (p. 1556) objects by means of the `source_timestamp` attribute inside the **DDS_SampleInfo** (p. 1344).*
- virtual **DDS_ReturnCode_t dispose_w_timestamp** (const char *key, const **DDS_InstanceHandle_t** &instance_handle, const **DDS_Time_t** &source_timestamp)

  *<<**extension**>> (p. 432) Performs the same functions as **DDSKeyedOctetsDataWriter::dispose** (p. 1761) except that the application provides the value for the `source_timestamp` that is made available to **DDSDataReader** (p. 1556) objects by means of the `source_timestamp` attribute inside the **DDS_SampleInfo** (p. 1344).*
- virtual **DDS_ReturnCode_t get_key_value** (**DDS_KeyedOctets** &key_holder, const **DDS_InstanceHandle_t** &handle)

  Retrieve the instance `key` that corresponds to an instance `handle`.
- virtual **DDS_ReturnCode_t get_key_value** (char *key, const **DDS_InstanceHandle_t** &handle)

  *<<**extension**>> (p. 432) Retrieve the instance `key` that corresponds to an instance `handle`.*
- virtual **DDS_InstanceHandle_t lookup_instance** (const **DDS_KeyedOctets** &key_holder)

  Retrieve the instance `handle` that corresponds to an instance `key_holder`.
- virtual **DDS_InstanceHandle_t lookup_instance** (const char *key)

  *<<**extension**>> (p. 432) Retrieve the instance `handle` that corresponds to an instance `key`.*

Static Public Member Functions

- static **DDSKeyedOctetsDataWriter** * **narrow** (**DDSDataWriter** *writer)

  *Narrow the given **DDSDataWriter** (p. 1589) pointer to a **DDSKeyedOctetsDataWriter** (p. 1750) pointer.*

9.263.1 Detailed Description

<<***interface***>> *(p. 433)* Instantiates `DataWriter` < **DDS_KeyedOctets** (p. 1168) >.

See also

FooDataWriter (p. 1944)

DDSDataWriter (p. 1589)

9.263.2 Member Function Documentation

9.263.2.1 narrow()

```
static DDSKeyedOctetsDataWriter * DDSKeyedOctetsDataWriter::narrow (
            DDSDataWriter * writer ) [static]
```

Narrow the given **DDSDataWriter** (p. 1589) pointer to a **DDSKeyedOctetsDataWriter** (p. 1750) pointer.

See also

FooDataWriter::narrow (p. 1946)

9.263.2.2 register_instance() [1/2]

```
virtual DDS_InstanceHandle_t DDSKeyedOctetsDataWriter::register_instance (
            const DDS_KeyedOctets & instance_data ) [virtual]
```

Informs RTI Connext that the application will be modifying a particular instance.

See also

FooDataWriter::register_instance (p. 1946)

9.263.2.3 register_instance() [2/2]

```
virtual  DDS_InstanceHandle_t DDSKeyedOctetsDataWriter::register_instance (
            const char * key )  [virtual]
```

<<***extension***>> *(p. 432)* Informs RTI Connext that the application will be modifying a particular instance.

See also

FooDataWriter::register_instance (p. 1946)

9.263.2.4 register_instance_w_timestamp() [1/2]

```
virtual  DDS_InstanceHandle_t DDSKeyedOctetsDataWriter::register_instance_w_timestamp (
            const  DDS_KeyedOctets & instance_data,
            const  DDS_Time_t & source_timestamp )  [virtual]
```

Performs the same functions as **DDSKeyedOctetsDataWriter::register_instance** (p. 1753) except that the application provides the value for the `source_timestamp`.

See also

FooDataWriter::register_instance_w_timestamp (p. 1947)

9.263.2.5 register_instance_w_timestamp() [2/2]

```
virtual  DDS_InstanceHandle_t DDSKeyedOctetsDataWriter::register_instance_w_timestamp (
            const char * key,
            const  DDS_Time_t & source_timestamp )  [virtual]
```

<<***extension***>> *(p. 432)* Performs the same functions as **DDSKeyedOctetsDataWriter::register_instance** (p. 1753) except that the application provides the value for the `source_timestamp`.

See also

FooDataWriter::register_instance_w_timestamp (p. 1947)

9.263.2.6 unregister_instance() [1/2]

```
virtual  DDS_ReturnCode_t DDSKeyedOctetsDataWriter::unregister_instance (
             const  DDS_KeyedOctets & instance_data,
             const  DDS_InstanceHandle_t & handle )  [virtual]
```

Reverses the action of **DDSKeyedOctetsDataWriter::register_instance** (p. 1753).

See also

FooDataWriter::unregister_instance (p. 1948)

9.263.2.7 unregister_instance() [2/2]

```
virtual  DDS_ReturnCode_t DDSKeyedOctetsDataWriter::unregister_instance (
             const char * key,
             const  DDS_InstanceHandle_t & handle )  [virtual]
```

<<***extension***>> *(p. 432)* Reverses the action of **DDSKeyedOctetsDataWriter::register_instance** (p. 1753).

See also

FooDataWriter::unregister_instance (p. 1948)

9.263.2.8 unregister_instance_w_timestamp() [1/2]

```
virtual  DDS_ReturnCode_t DDSKeyedOctetsDataWriter::unregister_instance_w_timestamp (
             const  DDS_KeyedOctets & instance_data,
             const  DDS_InstanceHandle_t & handle,
             const  DDS_Time_t & source_timestamp )  [virtual]
```

Performs the same function as **DDSKeyedOctetsDataWriter::unregister_instance** (p. 1754) except that it also provides the value for the `source_timestamp`.

See also

FooDataWriter::unregister_instance_w_timestamp (p. 1950)

9.263.2.9 unregister_instance_w_timestamp() [2/2]

```
virtual DDS_ReturnCode_t DDSKeyedOctetsDataWriter::unregister_instance_w_timestamp (
            const char * key,
            const DDS_InstanceHandle_t & handle,
            const DDS_Time_t & source_timestamp ) [virtual]
```

Performs the same function as **DDSKeyedOctetsDataWriter::unregister_instance** (p. 1754) except that it also provides the value for the `source_timestamp`.

See also

FooDataWriter::unregister_instance_w_timestamp (p. 1950)

9.263.2.10 create_data()

```
virtual DDS_KeyedOctets * DDSKeyedOctetsDataWriter::create_data (
            const DDS_TypeAllocationParams_t & alloc_params ) [virtual]
```

Creates a keyed octet sequence.

The size of the instance is determined by the DataWriter property **dds.builtin_type.keyed_octets.alloc_size**.

Default size: **dds.builtin_type.keyed_octets.max_size** property of DomainParticipant if defined. Otherwise 2048.

Created instances must be deleted with **DDSKeyedOctetsDataWriter::delete_data** (p. 1756).

Returns

Newly created keyed octet sequence, or NULL on failure.

See also

BuiltinTypeMemoryManagement section of Built-in Types.

9.263.2.11 delete_data()

```
virtual DDS_Boolean DDSKeyedOctetsDataWriter::delete_data (
             DDS_KeyedOctets * sample,
            const DDS_TypeDeallocationParams_t & dealloc_params ) [virtual]
```

Destroys a string data instance created by **DDSStringDataWriter::create_data** (p. 1853).

Returns

DDS_BOOLEAN_TRUE (p. 513) upon successful deletion.

9.263.2.12 write() [1/3]

```
virtual DDS_ReturnCode_t DDSKeyedOctetsDataWriter::write (
            const DDS_KeyedOctets & instance_data,
            const DDS_InstanceHandle_t & handle ) [virtual]
```

Modifies the value of a **DDS_KeyedOctets** (p. 1168) data instance.

See also

FooDataWriter::write (p. 1951)

9.263.2.13 write() [2/3]

```
virtual DDS_ReturnCode_t DDSKeyedOctetsDataWriter::write (
            const char * key,
            const unsigned char * octets,
            int length,
            const DDS_InstanceHandle_t & handle ) [virtual]
```

<<***extension***>> *(p. 432)* Modifies the value of a **DDS_KeyedOctets** (p. 1168) data instance.

Parameters

key	<<***in***>> *(p. 433)* Instance key.
octets	<<***in***>> *(p. 433)* Array of octets to be published.
length	<<***in***>> *(p. 433)* Number of octets to be published.
handle	<<***in***>> *(p. 433)* Either the handle returned by a previous call to **DDSKeyedOctetsDataWriter::register_instance** (p. 1753), or else the special value **DDS_HANDLE_NIL** (p. 78). See **FooDataWriter::write** (p. 1951).

See also

FooDataWriter::write (p. 1951)

9.263.2.14 write() [3/3]

```
virtual DDS_ReturnCode_t DDSKeyedOctetsDataWriter::write (
            const char * key,
            const DDS_OctetSeq & octets,
            const DDS_InstanceHandle_t & handle ) [virtual]
```

<<***extension***>> *(p. 432)* Modifies the value of a **DDS_KeyedOctets** (p. 1168) data instance.

Parameters

key	<<***in***>> *(p. 433)* Instance key.
octets	<<***in***>> *(p. 433)* Sequence of octets to be published.
handle	<<***in***>> *(p. 433)* Either the handle returned by a previous call to **DDSKeyedOctetsDataWriter::register_instance** (p. 1753), or else the special value **DDS_HANDLE_NIL** (p. 78). See **FooDataWriter::write** (p. 1951).

See also

FooDataWriter::write (p. 1951)

9.263.2.15 write_w_timestamp() [1/3]

```
virtual DDS_ReturnCode_t DDSKeyedOctetsDataWriter::write_w_timestamp (
            const DDS_KeyedOctets & instance_data,
            const DDS_InstanceHandle_t & handle,
            const DDS_Time_t & source_timestamp ) [virtual]
```

Performs the same function as **DDSKeyedOctetsDataWriter::write** (p. 1756) except that it also provides the value for the `source_timestamp`.

See also

FooDataWriter::write_w_timestamp (p. 1955)

9.263.2.16 write_w_timestamp() [2/3]

```
virtual DDS_ReturnCode_t DDSKeyedOctetsDataWriter::write_w_timestamp (
            const char * key,
            const unsigned char * octets,
            int length,
            const DDS_InstanceHandle_t & handle,
            const DDS_Time_t & source_timestamp ) [virtual]
```

Performs the same function as **DDSKeyedOctetsDataWriter::write** (p. 1756) except that it also provides the value for the `source_timestamp`.

Parameters

key	<<***in***>> *(p. 433)* Instance key.
octets	<<***in***>> *(p. 433)* Array of octets to be published.
length	<<***in***>> *(p. 433)* Number of octets to be published.
handle	<<***in***>> *(p. 433)* Either the handle returned by a previous call to **DDSKeyedOctetsDataWriter::register_instance** (p. 1753), or else the special value **DDS_HANDLE_NIL** (p. 78). See **FooDataWriter::write** (p. 1951).
source_timestamp	<<***in***>> *(p. 433)* The timestamp value must be greater than or equal to the timestamp value used in the last writer operation. See **FooDataWriter::write_w_timestamp** (p. 1955).

See also

FooDataWriter::write (p. 1951)

9.263.2.17 write_w_timestamp() [3/3]

```
virtual  DDS_ReturnCode_t DDSKeyedOctetsDataWriter::write_w_timestamp (
             const char * key,
             const  DDS_OctetSeq & octets,
             const  DDS_InstanceHandle_t & handle,
             const  DDS_Time_t & source_timestamp )  [virtual]
```

Performs the same function as **DDSKeyedOctetsDataWriter::write** (p. 1756) except that it also provides the value for the `source_timestamp`.

Parameters

key	<<***in***>> *(p. 433)* Instance key.
octets	<<***in***>> *(p. 433)* Sequence of octets to be published.
handle	<<***in***>> *(p. 433)* Either the handle returned by a previous call to **DDSKeyedOctetsDataWriter::register_instance** (p. 1753), or else the special value **DDS_HANDLE_NIL** (p. 78). See **FooDataWriter::write** (p. 1951).
source_timestamp	<<***in***>> *(p. 433)* The timestamp value must be greater than or equal to the timestamp value used in the last writer operation. See **FooDataWriter::write_w_timestamp** (p. 1955).

See also

FooDataWriter::write (p. 1951)

9.263.2.18 write_w_params() [1/3]

```
virtual  DDS_ReturnCode_t DDSKeyedOctetsDataWriter::write_w_params (
             const  DDS_KeyedOctets & instance_data,
              DDS_WriteParams_t & params )  [virtual]
```

Performs the same function as **DDSKeyedOctetsDataWriter::write** (p. 1756) except that it also allows specification of the instance handle, source timestamp, publication priority, and cookie.

See also

FooDataWriter::write_w_params (p. 1956)

9.263.2.19 write_w_params() [2/3]

```
virtual DDS_ReturnCode_t DDSKeyedOctetsDataWriter::write_w_params (
            const char * key,
            const unsigned char * octets,
            int length,
             DDS_WriteParams_t & params )  [virtual]
```

Performs the same function as **DDSKeyedOctetsDataWriter::write** (p. 1756) except that it also allows specification of the instance handle, source timestamp, publication priority, and cookie.

Parameters

key	<<***in***>> *(p. 433)* Instance key.
octets	<<***in***>> *(p. 433)* Array of octets to be published.
length	<<***in***>> *(p. 433)* Number of octets to be published.
params	<<***in***>> *(p. 433)* The **DDS_WriteParams_t** (p. 1519) parameter containing the instance handle, source timestamp, publication priority, and cookie to be used in write operation. See **FooDataWriter::write_w_params** (p. 1956).

See also

FooDataWriter::write_w_params (p. 1956)

9.263.2.20 write_w_params() [3/3]

```
virtual DDS_ReturnCode_t DDSKeyedOctetsDataWriter::write_w_params (
            const char * key,
            const  DDS_OctetSeq & octets,
             DDS_WriteParams_t & params )  [virtual]
```

Performs the same function as **DDSKeyedOctetsDataWriter::write** (p. 1756) except that it also allows specification of the instance handle, source timestamp, publication priority, and cookie.

Parameters

key	<<***in***>> *(p. 433)* Instance key.
octets	<<***in***>> *(p. 433)* Sequence of octets to be published.
params	<<***in***>> *(p. 433)* The **DDS_WriteParams_t** (p. 1519) parameter containing the instance handle, source timestamp, publication priority, and cookie to be used in write operation. See **FooDataWriter::write_w_params** (p. 1956).

See also

FooDataWriter::write_w_params (p. 1956)

9.263.2.21 dispose() [1/2]

```
virtual  DDS_ReturnCode_t DDSKeyedOctetsDataWriter::dispose (
             const  DDS_KeyedOctets & instance_data,
             const  DDS_InstanceHandle_t & handle )  [virtual]
```

Requests the middleware to delete the data.

See also

FooDataWriter::dispose (p. 1957)

9.263.2.22 dispose() [2/2]

```
virtual  DDS_ReturnCode_t DDSKeyedOctetsDataWriter::dispose (
             const char * key,
             const  DDS_InstanceHandle_t & instance_handle )  [virtual]
```

<<***extension***>> *(p. 432)* Requests the middleware to delete the data.

See also

FooDataWriter::dispose (p. 1957)

9.263.2.23 dispose_w_timestamp() [1/2]

```
virtual  DDS_ReturnCode_t DDSKeyedOctetsDataWriter::dispose_w_timestamp (
             const  DDS_KeyedOctets & instance_data,
             const  DDS_InstanceHandle_t & handle,
             const  DDS_Time_t & source_timestamp )  [virtual]
```

Performs the same functions as **DDSKeyedOctetsDataWriter::dispose** (p. 1761) except that the application provides the value for the `source_timestamp` that is made available to **DDSDataReader** (p. 1556) objects by means of the `source_timestamp` attribute inside the **DDS_SampleInfo** (p. 1344).

See also

FooDataWriter::dispose_w_timestamp (p. 1958)

9.263.2.24 dispose_w_timestamp() [2/2]

```
virtual  DDS_ReturnCode_t DDSKeyedOctetsDataWriter::dispose_w_timestamp (
             const char * key,
             const  DDS_InstanceHandle_t & instance_handle,
             const  DDS_Time_t & source_timestamp )  [virtual]
```

<<***extension***>> *(p. 432)* Performs the same functions as **DDSKeyedOctetsDataWriter::dispose** (p. 1761) except that the application provides the value for the `source_timestamp` that is made available to **DDSDataReader** (p. 1556) objects by means of the `source_timestamp` attribute inside the **DDS_SampleInfo** (p. 1344).

See also

FooDataWriter::dispose_w_timestamp (p. 1958)

9.263.2.25 get_key_value() [1/2]

```
virtual  DDS_ReturnCode_t DDSKeyedOctetsDataWriter::get_key_value (
              DDS_KeyedOctets & key_holder,
             const  DDS_InstanceHandle_t & handle )  [virtual]
```

Retrieve the instance `key` that corresponds to an instance `handle`.

See also

FooDataWriter::get_key_value (p. 1959)

9.263.2.26 get_key_value() [2/2]

```
virtual  DDS_ReturnCode_t DDSKeyedOctetsDataWriter::get_key_value (
             char * key,
             const  DDS_InstanceHandle_t & handle )  [virtual]
```

<<***extension***>> *(p. 432)* Retrieve the instance `key` that corresponds to an instance `handle`.

See also

FooDataWriter::get_key_value (p. 1959)

9.263.2.27 lookup_instance() [1/2]

```
virtual  DDS_InstanceHandle_t DDSKeyedOctetsDataWriter::lookup_instance (
            const  DDS_KeyedOctets & key_holder )  [virtual]
```

Retrieve the instance `handle` that corresponds to an instance `key_holder`.

See also

FooDataWriter::lookup_instance (p. 1960)

9.263.2.28 lookup_instance() [2/2]

```
virtual  DDS_InstanceHandle_t DDSKeyedOctetsDataWriter::lookup_instance (
            const char * key )  [virtual]
```

<<***extension***>> *(p. 432)* Retrieve the instance `handle` that corresponds to an instance `key`.

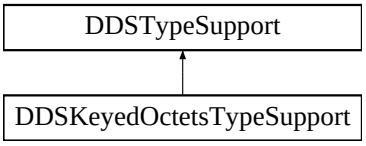
See also

FooDataWriter::lookup_instance (p. 1960)

9.264 DDSKeyedOctetsTypeSupport Class Reference

<<***interface***>> *(p. 433)* **DDS_KeyedOctets** (p. 1168) type support.

Inheritance diagram for DDSKeyedOctetsTypeSupport:



Static Public Member Functions

- static **DDS_ReturnCode_t** **register_type** (**DDSDomainParticipant** *participant, const char *type_name="DDS::KeyedOctets")

  *Allows an application to communicate to RTI Connext the existence of the **DDS_KeyedOctets** (p. 1168) data type.*
- static **DDS_ReturnCode_t** **unregister_type** (**DDSDomainParticipant** *participant, const char *type_name="DDS::KeyedOctets")

  *Allows an application to unregister the **DDS_KeyedOctets** (p. 1168) data type from RTI Connext. After calling unregister_type, no further communication using this type is possible.*
- static const char * **get_type_name** ()

  *Get the default name for the **DDS_KeyedOctets** (p. 1168) type.*
- static void **print_data** (const **DDS_KeyedOctets** *a_data)

  *<<**extension**>> (p. 432) Print value of data type to standard out.*
- static **DDS_TypeCode** * **get_typecode** ()

  *<<**extension**>> (p. 432) Retrieves the TypeCode for the Type.*
- static **DDS_ReturnCode_t** **serialize_data_to_cdr_buffer** (char *buffer, unsigned int &length, const **DDS_KeyedOctets** *a_data)

  *<<**extension**>> (p. 432) Serializes the input sample into a CDR buffer of octets.*
- static **DDS_ReturnCode_t** **serialize_data_to_cdr_buffer_ex** (char *buffer, unsigned int &length, const **DDS_KeyedOctets** *a_data, **DDS_DataRepresentationId_t** representation)

  *<<**extension**>> (p. 432) Serializes the input sample into a buffer of octets.*
- static **DDS_ReturnCode_t** **deserialize_data_from_cdr_buffer** (**DDS_KeyedOctets** *a_data, const char *buffer, unsigned int length)

  *<<**extension**>> (p. 432) Deserializes a sample from a buffer of octets.*
- static **DDS_ReturnCode_t** **data_to_string** (**DDS_KeyedOctets** *sample, char *str, **DDS_UnsignedLong** &str_size, const **DDS_PrintFormatProperty** &property)

  *<<**extension**>> (p. 432) Get the string representation of an input sample.*

9.264.1 Detailed Description

*<<**interface**>> (p. 433)* **DDS_KeyedOctets** (p. 1168) type support.

9.264.2 Member Function Documentation

9.264.2.1 register_type()

```
static DDS_ReturnCode_t DDSKeyedOctetsTypeSupport::register_type (
            DDSDomainParticipant * participant,
           const char * type_name = "DDS::KeyedOctets" )  [static]
```

Allows an application to communicate to RTI Connext the existence of the **DDS_KeyedOctets** (p. 1168) data type.

By default, The **DDS_KeyedOctets** (p. 1168) built-in type is automatically registered when a DomainParticipant is created using the type_name returned by **DDSKeyedOctetsTypeSupport::get_type_name** (p. 1767). Therefore, the usage of this function is optional and it is only required when the automatic built-in type registration is disabled using the participant property "dds.builtin_type.auto_register".

This method can also be used to register the same **DDSKeyedOctetsTypeSupport** (p. 1763) with a **DDSDomain↩ Participant** (p. 1619) using different values for the type_name.

If `register_type` is called multiple times with the same **DDSDomainParticipant** (p. 1619) and `type_name`, the second (and subsequent) registrations are ignored by the operation.

Parameters

participant	<<***in***>> *(*p. *433)* the **DDSDomainParticipant** (p. 1619) to register the data type **DDS_Octets** (p. 1217) with. Cannot be NULL.
type_name	<<***in***>> *(*p. *433)* the type name under with the data type **DDS_KeyedOctets** (p. 1168) is registered with the participant; this type name is used when creating a new **DDSTopic** (p. 1885). (See **DDSDomainParticipant::create_topic** (p. 1650).) The name may not be NULL or longer than 255 characters.

Exceptions

One	of the **Standard Return Codes** (p. 532), **DDS_RETCODE_PRECONDITION_NOT_MET** (p. 532) or **DDS_RETCODE_OUT_OF_RESOURCES** (p. 533).

MT Safety:

UNSAFE on the FIRST call. It is not safe for two threads to simultaneously make the first call to register a type. Subsequent calls are thread safe.

See also

DDSDomainParticipant::create_topic (p. 1650)

9.264.2.2 unregister_type()

```
static  DDS_ReturnCode_t DDSKeyedOctetsTypeSupport::unregister_type (
              DDSDomainParticipant * participant,
            const char * type_name = "DDS::KeyedOctets" )  [static]
```

Allows an application to unregister the **DDS_KeyedOctets** (p. 1168) data type from RTI Connext. After calling unregister_type, no further communication using this type is possible.

Precondition

The **DDS_KeyedOctets** (p. 1168) type with `type_name` is registered with the participant and all **DDSTopic** (p. 1885) objects referencing the type have been destroyed. If the type is not registered with the participant, or if any **DDSTopic** (p. 1885) is associated with the type, the operation will fail with **DDS_RETCODE_ERROR** (p. 532).

Postcondition

All information about the type is removed from RTI Connext. No further communication using this type is possible.

Parameters

participant	<<***in***>> *(p. 433)* the **DDSDomainParticipant** (p. 1619) to unregister the data type **DDS_KeyedOctets** (p. 1168) from. Cannot be NULL.
type_name	<<***in***>> *(p. 433)* the type name under with the data type **DDS_KeyedOctets** (p. 1168) is registered with the participant. The name should match a name that has been previously used to register a type with the participant. Cannot be NULL.

Exceptions

One	of the **Standard Return Codes** (p. 532), **DDS_RETCODE_BAD_PARAMETER** (p. 532) or **DDS_RETCODE_ERROR** (p. 532)

MT Safety:

SAFE.

See also

DDSKeyedOctetsTypeSupport::register_type (p. 1764)

9.264.2.3 get_type_name()

```
static const char * DDSKeyedOctetsTypeSupport::get_type_name ( )  [static]
```

Get the default name for the **DDS_KeyedOctets** (p. 1168) type.

Can be used for calling **DDSKeyedOctetsTypeSupport::register_type** (p. 1764) or creating **DDSTopic** (p. 1885).

Returns

default name for the **DDS_KeyedOctets** (p. 1168) type.

See also

DDSKeyedOctetsTypeSupport::register_type (p. 1764)

DDSDomainParticipant::create_topic (p. 1650)

9.264.2.4 print_data()

```
static void DDSKeyedOctetsTypeSupport::print_data (
            const  DDS_KeyedOctets * a_data )  [static]
```

<<***extension***>> *(p. 432)* Print value of data type to standard out.

The *generated* implementation of the operation knows how to print value of a data type.

Parameters

a_data	<<***in***>> *(p. 433)* **DDS_KeyedOctets** (p. 1168) to be printed.

9.264.2.5 get_typecode()

```
static  DDS_TypeCode * DDSKeyedOctetsTypeSupport::get_typecode ( )  [static]
```

<<***extension***>> *(p. 432)* Retrieves the TypeCode for the Type.

See also

FooTypeSupport::get_typecode (p. 1991)

9.264.2.6 serialize_data_to_cdr_buffer()

```
static  DDS_ReturnCode_t DDSKeyedOctetsTypeSupport::serialize_data_to_cdr_buffer (
             char * buffer,
             unsigned int & length,
             const  DDS_KeyedOctets * a_data )  [static]
```

<<***extension***>> *(p. 432)* Serializes the input sample into a CDR buffer of octets.

See also

FooTypeSupport::serialize_data_to_cdr_buffer (p. 1989)

9.264.2.7 serialize_data_to_cdr_buffer_ex()

```
static  DDS_ReturnCode_t DDSKeyedOctetsTypeSupport::serialize_data_to_cdr_buffer_ex (
             char * buffer,
             unsigned int & length,
             const  DDS_KeyedOctets * a_data,
              DDS_DataRepresentationId_t representation )  [static]
```

<<***extension***>> *(p. 432)* Serializes the input sample into a buffer of octets.

See also

FooTypeSupport::serialize_data_to_cdr_buffer (p. 1989)

9.264.2.8 deserialize_data_from_cdr_buffer()

```
static DDS_ReturnCode_t DDSKeyedOctetsTypeSupport::deserialize_data_from_cdr_buffer (
            DDS_KeyedOctets * a_data,
            const char * buffer,
            unsigned int length )  [static]
```

<<***extension***>> *(p. 432)* Deserializes a sample from a buffer of octets.

See also

FooTypeSupport::deserialize_data_from_cdr_buffer (p. 1990)

9.264.2.9 data_to_string()

```
static DDS_ReturnCode_t DDSKeyedOctetsTypeSupport::data_to_string (
            DDS_KeyedOctets * sample,
            char * str,
            DDS_UnsignedLong & str_size,
            const DDS_PrintFormatProperty & property )  [static]
```

<<***extension***>> *(p. 432)* Get the string representation of an input sample.

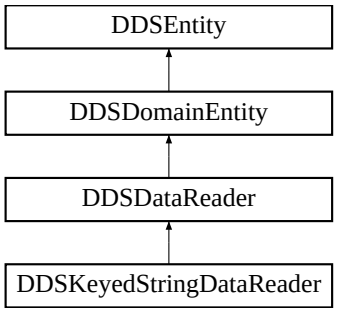
See also

FooTypeSupport::data_to_string (p. 1991)

9.265 DDSKeyedStringDataReader Class Reference

<<***interface***>> *(p. 433)* Instantiates `DataReader` < **DDS_KeyedString** (p. 1171) >.

Inheritance diagram for DDSKeyedStringDataReader:



Public Member Functions

- virtual **DDS_ReturnCode_t read** (**DDS_KeyedStringSeq** &received_data, **DDS_SampleInfoSeq** &info_seq, **DDS_Long** max_samples= **DDS_LENGTH_UNLIMITED**, **DDS_SampleStateMask** sample_states= **DDS_ANY_SAMPLE_STATE**, **DDS_ViewStateMask** view_states= **DDS_ANY_VIEW_STATE**, **DDS_InstanceStateMask** instance_states= **DDS_ANY_INSTANCE_STATE**)

  *Access a collection of data samples from the **DDSDataReader** (*p. *1556).*

- virtual **DDS_ReturnCode_t take** (**DDS_KeyedStringSeq** &received_data, **DDS_SampleInfoSeq** &info_seq, **DDS_Long** max_samples= **DDS_LENGTH_UNLIMITED**, **DDS_SampleStateMask** sample_states= **DDS_ANY_SAMPLE_STATE**, **DDS_ViewStateMask** view_states= **DDS_ANY_VIEW_STATE**, **DDS_InstanceStateMask** instance_states= **DDS_ANY_INSTANCE_STATE**)

  *Access a collection of data-samples from the **DDSDataReader** (*p. *1556).*

- virtual **DDS_ReturnCode_t read_w_condition** (**DDS_KeyedStringSeq** &received_data, **DDS_SampleInfoSeq** &info_seq, **DDS_Long** max_samples, **DDSReadCondition** ∗condition)

  *Accesses via **DDSKeyedStringDataReader::read** (*p. *1772) the samples that match the criteria specified in the **DDSReadCondition** (*p. *1842).*

- virtual **DDS_ReturnCode_t take_w_condition** (**DDS_KeyedStringSeq** &received_data, **DDS_SampleInfoSeq** &info_seq, **DDS_Long** max_samples, **DDSReadCondition** ∗condition)

  *Analogous to **DDSKeyedStringDataReader::read_w_condition** (*p. *1772) except it accesses samples via the **DDSKeyedStringDataReader::take** (*p. *1772) operation.*

- virtual **DDS_ReturnCode_t read_next_sample** (**DDS_KeyedString** &received_data, **DDS_SampleInfo** &sample_info)

  *Copies the next not-previously-accessed data value from the **DDSDataReader** (*p. *1556).*

- virtual **DDS_ReturnCode_t take_next_sample** (**DDS_KeyedString** &received_data, **DDS_SampleInfo** &sample_info)

  *Copies the next not-previously-accessed data value from the **DDSDataReader** (*p. *1556).*

- virtual **DDS_ReturnCode_t read_instance** (**DDS_KeyedStringSeq** &received_data, **DDS_SampleInfoSeq** &info_seq, **DDS_Long** max_samples= **DDS_LENGTH_UNLIMITED**, const **DDS_InstanceHandle_t** &a_handle= **DDS_HANDLE_NIL**, **DDS_SampleStateMask** sample_states= **DDS_ANY_SAMPLE_STATE**, **DDS_ViewStateMask** view_states= **DDS_ANY_VIEW_STATE**, **DDS_InstanceStateMask** instance_states= **DDS_ANY_INSTANCE_STATE**)

  *Access a collection of data samples from the **DDSDataReader** (*p. *1556).*

- virtual **DDS_ReturnCode_t take_instance** (**DDS_KeyedStringSeq** &received_data, **DDS_SampleInfoSeq** &info_seq, **DDS_Long** max_samples= **DDS_LENGTH_UNLIMITED**, const **DDS_InstanceHandle_t** &a_handle= **DDS_HANDLE_NIL**, **DDS_SampleStateMask** sample_states= **DDS_ANY_SAMPLE_STATE**, **DDS_ViewStateMask** view_states= **DDS_ANY_VIEW_STATE**, **DDS_InstanceStateMask** instance_states= **DDS_ANY_INSTANCE_STATE**)

  *Access a collection of data samples from the **DDSDataReader** (*p. *1556).*

- virtual **DDS_ReturnCode_t read_instance_w_condition** (**DDS_KeyedStringSeq** &received_data, **DDS_SampleInfoSeq** &info_seq, **DDS_Long** max_samples= **DDS_LENGTH_UNLIMITED**, const **DDS_InstanceHandle_t** &a_handle= **DDS_HANDLE_NIL**, **DDSReadCondition** ∗condition=NULL)

  *Accesses via **DDSKeyedStringDataReader::read_instance** (*p. *1773) the samples that match the criteria specified in the **DDSReadCondition** (*p. *1842).*

- virtual **DDS_ReturnCode_t take_instance_w_condition** (**DDS_KeyedStringSeq** &received_data, **DDS_SampleInfoSeq** &info_seq, **DDS_Long** max_samples= **DDS_LENGTH_UNLIMITED**, const **DDS_InstanceHandle_t** &a_handle= **DDS_HANDLE_NIL**, **DDSReadCondition** ∗condition=NULL)

  *Accesses via **DDSKeyedStringDataReader::take_instance** (*p. *1774) the samples that match the criteria specified in the **DDSReadCondition** (*p. *1842).*

- virtual **DDS_ReturnCode_t read_next_instance** (**DDS_KeyedStringSeq** &received_data, **DDS_SampleInfoSeq** &info_seq, **DDS_Long** max_samples= **DDS_LENGTH_UNLIMITED**, const **DDS_InstanceHandle_t** &previous_handle= **DDS_HANDLE_NIL**, **DDS_SampleStateMask** sample_states= **DDS_ANY_SAMPLE_STATE**, **DDS_ViewStateMask** view_states= **DDS_ANY_VIEW_STATE**, **DDS_InstanceStateMask** instance_states= **DDS_ANY_INSTANCE_STATE**)

*Access a collection of data samples from the **DDSDataReader*** (p. *1556*).

- virtual **DDS_ReturnCode_t** **take_next_instance** (**DDS_KeyedStringSeq** &received_data, **DDS_SampleInfoSeq** &info_seq, **DDS_Long** max_samples= **DDS_LENGTH_UNLIMITED**, const **DDS_InstanceHandle_t** &previous_handle= **DDS_HANDLE_NIL**, **DDS_SampleStateMask** sample_states= **DDS_ANY_SAMPLE_STATE**, **DDS_ViewStateMask** view_states= **DDS_ANY_VIEW_STATE**, **DDS_InstanceStateMask** instance_states= **DDS_ANY_INSTANCE_STATE**)

  *Access a collection of data samples from the **DDSDataReader*** (p. *1556*).
- virtual **DDS_ReturnCode_t** **read_next_instance_w_condition** (**DDS_KeyedStringSeq** &received_data, **DDS_SampleInfoSeq** &info_seq, **DDS_Long** max_samples= **DDS_LENGTH_UNLIMITED**, const **DDS_InstanceHandle_t** &previous_handle= **DDS_HANDLE_NIL**, **DDSReadCondition** *condition=NULL)

  *Accesses via **DDSKeyedStringDataReader::read_next_instance*** (p. *1775*) *the samples that match the criteria specified in the **DDSReadCondition*** (p. *1842*).
- virtual **DDS_ReturnCode_t** **take_next_instance_w_condition** (**DDS_KeyedStringSeq** &received_data, **DDS_SampleInfoSeq** &info_seq, **DDS_Long** max_samples= **DDS_LENGTH_UNLIMITED**, const **DDS_InstanceHandle_t** &previous_handle= **DDS_HANDLE_NIL**, **DDSReadCondition** *condition=NULL)

  *Accesses via **DDSKeyedStringDataReader::take_next_instance*** (p. *1775*) *the samples that match the criteria specified in the **DDSReadCondition*** (p. *1842*).
- virtual **DDS_ReturnCode_t** **return_loan** (**DDS_KeyedStringSeq** &received_data, **DDS_SampleInfoSeq** &info_seq)

  *Indicates to the **DDSDataReader*** (p. *1556*) *that the application is done accessing the collection of* `received_data` *and* `info_seq` *obtained by some earlier invocation of read or take on the **DDSDataReader*** (p. *1556*).
- virtual **DDS_ReturnCode_t** **get_key_value** (**DDS_KeyedString** &key_holder, const **DDS_InstanceHandle_t** &handle)

  Retrieve the instance `key` *that corresponds to an instance* `handle`.
- virtual **DDS_ReturnCode_t** **get_key_value** (char *key, const **DDS_InstanceHandle_t** &handle)

  <<***extension***>> (p. *432*) *Retrieve the instance* `key` *that corresponds to an instance* `handle`.
- virtual **DDS_InstanceHandle_t** **lookup_instance** (const **DDS_KeyedString** &key_holder)

  Retrieve the instance `handle` *that corresponds to an instance* `key_holder`.
- virtual **DDS_InstanceHandle_t** **lookup_instance** (const char *key)

  <<***extension***>> (p. *432*) *Retrieve the instance* `handle` *that corresponds to an instance* `key`.

Static Public Member Functions

- static **DDSKeyedStringDataReader** * **narrow** (**DDSDataReader** *reader)

  *Narrow the given **DDSDataReader*** (p. *1556*) *pointer to a **DDSKeyedStringDataReader*** (p. *1769*) *pointer.*

9.265.1 Detailed Description

<<***interface***>> (p. *433*) Instantiates `DataReader` < **DDS_KeyedString** (p. 1171) >.

When reading or taking data with this reader, if you request a copy of the samples instead of a loan, and a string in a destination data sample is NULL, the middleware will allocate a new string for you of sufficient length to hold the received string. The new string will be allocated with **DDS_String_alloc** (p. 759); the sample's destructor will delete it.

A non- NULL string is assumed to be allocated to sufficient length to store the incoming data. It will not be reallocated.

See also

FooDataReader (p. 1917)

DDSDataReader (p. 1556)

9.265.2 Member Function Documentation

9.265.2.1 read()

```
virtual DDS_ReturnCode_t DDSKeyedStringDataReader::read (
            DDS_KeyedStringSeq & received_data,
            DDS_SampleInfoSeq & info_seq,
            DDS_Long max_samples = DDS_LENGTH_UNLIMITED,
            DDS_SampleStateMask sample_states = DDS_ANY_SAMPLE_STATE,
            DDS_ViewStateMask view_states = DDS_ANY_VIEW_STATE,
            DDS_InstanceStateMask instance_states = DDS_ANY_INSTANCE_STATE ) [virtual]
```

Access a collection of data samples from the **DDSDataReader** (p. 1556).

See also

FooDataReader::read (p. 1919)

9.265.2.2 take()

```
virtual DDS_ReturnCode_t DDSKeyedStringDataReader::take (
            DDS_KeyedStringSeq & received_data,
            DDS_SampleInfoSeq & info_seq,
            DDS_Long max_samples = DDS_LENGTH_UNLIMITED,
            DDS_SampleStateMask sample_states = DDS_ANY_SAMPLE_STATE,
            DDS_ViewStateMask view_states = DDS_ANY_VIEW_STATE,
            DDS_InstanceStateMask instance_states = DDS_ANY_INSTANCE_STATE ) [virtual]
```

Access a collection of data-samples from the **DDSDataReader** (p. 1556).

See also

FooDataReader::take (p. 1920)

9.265.2.3 read_w_condition()

```
virtual DDS_ReturnCode_t DDSKeyedStringDataReader::read_w_condition (
            DDS_KeyedStringSeq & received_data,
            DDS_SampleInfoSeq & info_seq,
            DDS_Long max_samples,
            DDSReadCondition * condition ) [virtual]
```

Accesses via **DDSKeyedStringDataReader::read** (p. 1772) the samples that match the criteria specified in the **DDSReadCondition** (p. 1842).

See also

FooDataReader::read_w_condition (p. 1925)

9.265.2.4 take_w_condition()

```
virtual DDS_ReturnCode_t DDSKeyedStringDataReader::take_w_condition (
            DDS_KeyedStringSeq & received_data,
            DDS_SampleInfoSeq & info_seq,
            DDS_Long max_samples,
            DDSReadCondition * condition ) [virtual]
```

Analogous to **DDSKeyedStringDataReader::read_w_condition** (p. 1772) except it accesses samples via the **DDSKeyedStringDataReader::take** (p. 1772) operation.

See also

FooDataReader::take_w_condition (p. 1926)

9.265.2.5 read_next_sample()

```
virtual DDS_ReturnCode_t DDSKeyedStringDataReader::read_next_sample (
            DDS_KeyedString & received_data,
            DDS_SampleInfo & sample_info ) [virtual]
```

Copies the next not-previously-accessed data value from the **DDSDataReader** (p. 1556).

See also

FooDataReader::read_next_sample (p. 1927)

9.265.2.6 take_next_sample()

```
virtual DDS_ReturnCode_t DDSKeyedStringDataReader::take_next_sample (
            DDS_KeyedString & received_data,
            DDS_SampleInfo & sample_info ) [virtual]
```

Copies the next not-previously-accessed data value from the **DDSDataReader** (p. 1556).

See also

FooDataReader::take_next_sample (p. 1928)

9.265.2.7 read_instance()

```
virtual DDS_ReturnCode_t DDSKeyedStringDataReader::read_instance (
            DDS_KeyedStringSeq & received_data,
            DDS_SampleInfoSeq & info_seq,
            DDS_Long max_samples = DDS_LENGTH_UNLIMITED,
           const DDS_InstanceHandle_t & a_handle = DDS_HANDLE_NIL,
            DDS_SampleStateMask sample_states = DDS_ANY_SAMPLE_STATE,
            DDS_ViewStateMask view_states = DDS_ANY_VIEW_STATE,
            DDS_InstanceStateMask instance_states = DDS_ANY_INSTANCE_STATE ) [virtual]
```

Access a collection of data samples from the **DDSDataReader** (p. 1556).

See also

FooDataReader::read_instance (p. 1929)

9.265.2.8 take_instance()

```
virtual DDS_ReturnCode_t DDSKeyedStringDataReader::take_instance (
            DDS_KeyedStringSeq & received_data,
            DDS_SampleInfoSeq & info_seq,
            DDS_Long max_samples = DDS_LENGTH_UNLIMITED,
           const DDS_InstanceHandle_t & a_handle = DDS_HANDLE_NIL,
            DDS_SampleStateMask sample_states = DDS_ANY_SAMPLE_STATE,
            DDS_ViewStateMask view_states = DDS_ANY_VIEW_STATE,
            DDS_InstanceStateMask instance_states = DDS_ANY_INSTANCE_STATE ) [virtual]
```

Access a collection of data samples from the **DDSDataReader** (p. 1556).

See also

FooDataReader::take_instance (p. 1931)

9.265.2.9 read_instance_w_condition()

```
virtual DDS_ReturnCode_t DDSKeyedStringDataReader::read_instance_w_condition (
            DDS_KeyedStringSeq & received_data,
            DDS_SampleInfoSeq & info_seq,
            DDS_Long max_samples = DDS_LENGTH_UNLIMITED,
           const DDS_InstanceHandle_t & a_handle = DDS_HANDLE_NIL,
            DDSReadCondition * condition = NULL ) [virtual]
```

Accesses via **DDSKeyedStringDataReader::read_instance** (p. 1773) the samples that match the criteria specified in the **DDSReadCondition** (p. 1842).

See also

FooDataReader::read_instance_w_condition (p. 1932)

9.265.2.10 take_instance_w_condition()

```
virtual  DDS_ReturnCode_t DDSKeyedStringDataReader::take_instance_w_condition (
              DDS_KeyedStringSeq & received_data,
              DDS_SampleInfoSeq & info_seq,
              DDS_Long max_samples =  DDS_LENGTH_UNLIMITED,
             const  DDS_InstanceHandle_t & a_handle =  DDS_HANDLE_NIL,
              DDSReadCondition * condition = NULL )  [virtual]
```

Accesses via **DDSKeyedStringDataReader::take_instance** (p. 1774) the samples that match the criteria specified in the **DDSReadCondition** (p. 1842).

See also

FooDataReader::take_instance_w_condition (p. 1933)

9.265.2.11 read_next_instance()

```
virtual  DDS_ReturnCode_t DDSKeyedStringDataReader::read_next_instance (
              DDS_KeyedStringSeq & received_data,
              DDS_SampleInfoSeq & info_seq,
              DDS_Long max_samples =  DDS_LENGTH_UNLIMITED,
             const  DDS_InstanceHandle_t & previous_handle =  DDS_HANDLE_NIL,
              DDS_SampleStateMask sample_states =  DDS_ANY_SAMPLE_STATE,
              DDS_ViewStateMask view_states =  DDS_ANY_VIEW_STATE,
              DDS_InstanceStateMask instance_states =  DDS_ANY_INSTANCE_STATE )  [virtual]
```

Access a collection of data samples from the **DDSDataReader** (p. 1556).

See also

FooDataReader::read_next_instance (p. 1935)

9.265.2.12 take_next_instance()

```
virtual  DDS_ReturnCode_t DDSKeyedStringDataReader::take_next_instance (
              DDS_KeyedStringSeq & received_data,
              DDS_SampleInfoSeq & info_seq,
              DDS_Long max_samples =  DDS_LENGTH_UNLIMITED,
             const  DDS_InstanceHandle_t & previous_handle =  DDS_HANDLE_NIL,
              DDS_SampleStateMask sample_states =  DDS_ANY_SAMPLE_STATE,
              DDS_ViewStateMask view_states =  DDS_ANY_VIEW_STATE,
              DDS_InstanceStateMask instance_states =  DDS_ANY_INSTANCE_STATE )  [virtual]
```

Access a collection of data samples from the **DDSDataReader** (p. 1556).

See also

FooDataReader::take_next_instance (p. 1936)

9.265.2.13 read_next_instance_w_condition()

```
virtual DDS_ReturnCode_t DDSKeyedStringDataReader::read_next_instance_w_condition (
            DDS_KeyedStringSeq & received_data,
            DDS_SampleInfoSeq & info_seq,
            DDS_Long max_samples = DDS_LENGTH_UNLIMITED,
           const DDS_InstanceHandle_t & previous_handle = DDS_HANDLE_NIL,
            DDSReadCondition * condition = NULL ) [virtual]
```

Accesses via **DDSKeyedStringDataReader::read_next_instance** (p. 1775) the samples that match the criteria specified in the **DDSReadCondition** (p. 1842).

See also

FooDataReader::read_next_instance_w_condition (p. 1938)

9.265.2.14 take_next_instance_w_condition()

```
virtual DDS_ReturnCode_t DDSKeyedStringDataReader::take_next_instance_w_condition (
            DDS_KeyedStringSeq & received_data,
            DDS_SampleInfoSeq & info_seq,
            DDS_Long max_samples = DDS_LENGTH_UNLIMITED,
           const DDS_InstanceHandle_t & previous_handle = DDS_HANDLE_NIL,
            DDSReadCondition * condition = NULL ) [virtual]
```

Accesses via **DDSKeyedStringDataReader::take_next_instance** (p. 1775) the samples that match the criteria specified in the **DDSReadCondition** (p. 1842).

See also

FooDataReader::take_next_instance_w_condition (p. 1939)

9.265.2.15 return_loan()

```
virtual DDS_ReturnCode_t DDSKeyedStringDataReader::return_loan (
            DDS_KeyedStringSeq & received_data,
            DDS_SampleInfoSeq & info_seq ) [virtual]
```

Indicates to the **DDSDataReader** (p. 1556) that the application is done accessing the collection of `received_data` and `info_seq` obtained by some earlier invocation of read or take on the **DDSDataReader** (p. 1556).

See also

FooDataReader::return_loan (p. 1940)

9.265.2.16 get_key_value() [1/2]

```
virtual  DDS_ReturnCode_t DDSKeyedStringDataReader::get_key_value (
             DDS_KeyedString & key_holder,
            const  DDS_InstanceHandle_t & handle )  [virtual]
```

Retrieve the instance `key` that corresponds to an instance `handle`.

See also

FooDataReader::get_key_value (p. 1941)

9.265.2.17 get_key_value() [2/2]

```
virtual  DDS_ReturnCode_t DDSKeyedStringDataReader::get_key_value (
            char * key,
            const  DDS_InstanceHandle_t & handle )  [virtual]
```

<<***extension***>> *(p. 432)* Retrieve the instance `key` that corresponds to an instance `handle`.

See also

FooDataReader::get_key_value (p. 1941)

9.265.2.18 lookup_instance() [1/2]

```
virtual  DDS_InstanceHandle_t DDSKeyedStringDataReader::lookup_instance (
            const  DDS_KeyedString & key_holder )  [virtual]
```

Retrieve the instance `handle` that corresponds to an instance `key_holder`.

See also

FooDataReader::lookup_instance (p. 1942)

9.265.2.19 lookup_instance() [2/2]

```
virtual DDS_InstanceHandle_t DDSKeyedStringDataReader::lookup_instance (
            const char * key ) [virtual]
```

<<***extension***>> *(p. 432)* Retrieve the instance `handle` that corresponds to an instance `key`.

See also

FooDataReader::lookup_instance (p. 1942)

9.265.2.20 narrow()

```
static DDSKeyedStringDataReader * DDSKeyedStringDataReader::narrow (
             DDSDataReader * reader ) [static]
```

Narrow the given **DDSDataReader** (p. 1556) pointer to a **DDSKeyedStringDataReader** (p. 1769) pointer.

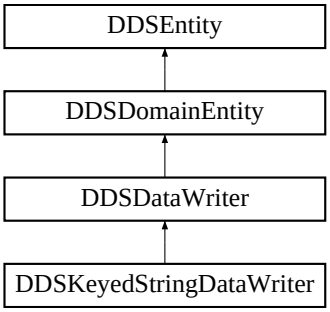
See also

FooDataReader::narrow (p. 1919)

9.266 DDSKeyedStringDataWriter Class Reference

<<***interface***>> *(p. 433)* Instantiates `DataWriter` < **DDS_KeyedString** (p. 1171) >.

Inheritance diagram for DDSKeyedStringDataWriter:



Public Member Functions

- virtual **DDS_InstanceHandle_t register_instance** (const **DDS_KeyedString** &instance_data)

  Informs RTI Connext that the application will be modifying a particular instance.
- virtual **DDS_InstanceHandle_t register_instance** (const char *key)

  *<<**extension**>> (p. 432) Informs RTI Connext that the application will be modifying a particular instance.*
- virtual **DDS_InstanceHandle_t register_instance_w_timestamp** (const **DDS_KeyedString** &instance_data, const **DDS_Time_t** &source_timestamp)

  *Performs the same functions as **DDSKeyedStringDataWriter::register_instance** (p. 1781) except that the application provides the value for the `source_timestamp`.*
- virtual **DDS_InstanceHandle_t register_instance_w_timestamp** (const char *key, const **DDS_Time_t** &source_timestamp)

  *<<**extension**>> (p. 432) Performs the same functions as **DDSKeyedStringDataWriter::register_instance** (p. 1781) except that the application provides the value for the `source_timestamp`.*
- virtual **DDS_ReturnCode_t unregister_instance** (const **DDS_KeyedString** &instance_data, const **DDS_InstanceHandle_t** &handle)

  *Reverses the action of **DDSKeyedStringDataWriter::register_instance** (p. 1781).*
- virtual **DDS_ReturnCode_t unregister_instance** (const char *key, const **DDS_InstanceHandle_t** &handle)

  *<<**extension**>> (p. 432) Reverses the action of **DDSKeyedStringDataWriter::register_instance** (p. 1781).*
- virtual **DDS_ReturnCode_t unregister_instance_w_timestamp** (const **DDS_KeyedString** &instance_data, const **DDS_InstanceHandle_t** &handle, const **DDS_Time_t** &source_timestamp)

  *Performs the same function as **DDSKeyedStringDataWriter::unregister_instance** (p. 1782) except that it also provides the value for the `source_timestamp`.*
- virtual **DDS_ReturnCode_t unregister_instance_w_timestamp** (const char *key, const **DDS_InstanceHandle_t** &handle, const **DDS_Time_t** &source_timestamp)

  *<<**extension**>> (p. 432) Performs the same function as **DDSKeyedStringDataWriter::unregister_instance** (p. 1782) except that it also provides the value for the `source_timestamp`.*
- virtual **DDS_KeyedString** * **create_data** (const **DDS_TypeAllocationParams_t** &alloc_params)

  Creates a keyed string data instance.
- virtual **DDS_Boolean delete_data** (**DDS_KeyedString** *sample, const **DDS_TypeDeallocationParams_t** &dealloc_params)

  *Destroys a keyed string data instance created by **DDSKeyedStringDataWriter::create_data** (p. 1783).*
- virtual **DDS_ReturnCode_t write** (const **DDS_KeyedString** &instance_data, const **DDS_InstanceHandle_t** &handle)

  *Modifies the value of a **DDS_KeyedString** (p. 1171) data instance.*
- virtual **DDS_ReturnCode_t write** (const char *key, const char *str, const **DDS_InstanceHandle_t** &handle)

  *<<**extension**>> (p. 432) Modifies the value of a **DDS_KeyedString** (p. 1171) data instance.*
- virtual **DDS_ReturnCode_t write_w_timestamp** (const **DDS_KeyedString** &instance_data, const **DDS_InstanceHandle_t** &handle, const **DDS_Time_t** &source_timestamp)

  *Performs the same function as **DDSKeyedStringDataWriter::write** (p. 1784) except that it also provides the value for the `source_timestamp`.*
- virtual **DDS_ReturnCode_t write_w_timestamp** (const char *key, const char *str, const **DDS_InstanceHandle_t** &handle, const **DDS_Time_t** &source_timestamp)

  *<<**extension**>> (p. 432) Performs the same function as **DDSKeyedStringDataWriter::write** (p. 1784) except that it also provides the value for the `source_timestamp`.*
- virtual **DDS_ReturnCode_t write_w_params** (const **DDS_KeyedString** &instance_data, **DDS_WriteParams_t** &params)

  *Performs the same function as **DDSKeyedStringDataWriter::write** (p. 1784) except that it also allows specification of the instance handle, source timestamp, publication priority, and cookie.*

- virtual **DDS_ReturnCode_t write_w_params** (const char *key, const char *str, **DDS_WriteParams_t** &params)

  *<<**extension**>> (p. 432) Performs the same function as **DDSKeyedStringDataWriter::write** (p. 1784) except that it also allows specification of the instance handle, source timestamp, publication priority, and cookie.*
- virtual **DDS_ReturnCode_t dispose** (const **DDS_KeyedString** &instance_data, const **DDS_InstanceHandle_t** &handle)

  Requests the middleware to delete the data.
- virtual **DDS_ReturnCode_t dispose** (const char *key, const **DDS_InstanceHandle_t** &instance_handle)

  *<<**extension**>> (p. 432) Requests the middleware to delete the data.*
- virtual **DDS_ReturnCode_t dispose_w_timestamp** (const **DDS_KeyedString** &instance_data, const **DDS_InstanceHandle_t** &handle, const **DDS_Time_t** &source_timestamp)

  *Performs the same functions as **DDSKeyedStringDataWriter::dispose** (p. 1786) except that the application provides the value for the `source_timestamp` that is made available to **DDSDataReader** (p. 1556) objects by means of the `source_timestamp` attribute inside the **DDS_SampleInfo** (p. 1344).*
- virtual **DDS_ReturnCode_t dispose_w_timestamp** (const char *key, const **DDS_InstanceHandle_t** &instance_handle, const **DDS_Time_t** &source_timestamp)

  *<<**extension**>> (p. 432) Performs the same functions as **DDSKeyedStringDataWriter::dispose** (p. 1786) except that the application provides the value for the `source_timestamp` that is made available to **DDSDataReader** (p. 1556) objects by means of the `source_timestamp` attribute inside the **DDS_SampleInfo** (p. 1344).*
- virtual **DDS_ReturnCode_t get_key_value** (**DDS_KeyedString** &key_holder, const **DDS_InstanceHandle_t** &handle)

  Retrieve the instance `key` that corresponds to an instance `handle`.
- virtual **DDS_ReturnCode_t get_key_value** (char *key, const **DDS_InstanceHandle_t** &handle)

  *<<**extension**>> (p. 432) Retrieve the instance `key` that corresponds to an instance `handle`.*
- virtual **DDS_InstanceHandle_t lookup_instance** (const **DDS_KeyedString** &key_holder)

  Retrieve the instance `handle` that corresponds to an instance `key_holder`.
- virtual **DDS_InstanceHandle_t lookup_instance** (const char *key)

  *<<**extension**>> (p. 432) Retrieve the instance `handle` that corresponds to an instance `key`.*

Static Public Member Functions

- static **DDSKeyedStringDataWriter** * **narrow** (**DDSDataWriter** *writer)

  *Narrow the given **DDSDataWriter** (p. 1589) pointer to a **DDSKeyedStringDataWriter** (p. 1778) pointer.*

9.266.1 Detailed Description

*<<**interface**>> (p. 433)* Instantiates `DataWriter` < **DDS_KeyedString** (p. 1171) >.

See also

FooDataWriter (p. 1944)

DDSDataWriter (p. 1589)

9.266.2 Member Function Documentation

9.266.2.1 narrow()

```
static DDSKeyedStringDataWriter * DDSKeyedStringDataWriter::narrow (
            DDSDataWriter * writer ) [static]
```

Narrow the given **DDSDataWriter** (p. 1589) pointer to a **DDSKeyedStringDataWriter** (p. 1778) pointer.

See also

FooDataWriter::narrow (p. 1946)

9.266.2.2 register_instance() [1/2]

```
virtual DDS_InstanceHandle_t DDSKeyedStringDataWriter::register_instance (
            const DDS_KeyedString & instance_data ) [virtual]
```

Informs RTI Connext that the application will be modifying a particular instance.

See also

FooDataWriter::register_instance (p. 1946)

9.266.2.3 register_instance() [2/2]

```
virtual DDS_InstanceHandle_t DDSKeyedStringDataWriter::register_instance (
            const char * key ) [virtual]
```

<<***extension***>> *(p. 432)* Informs RTI Connext that the application will be modifying a particular instance.

See also

FooDataWriter::register_instance (p. 1946)

9.266.2.4 register_instance_w_timestamp() [1/2]

```
virtual DDS_InstanceHandle_t DDSKeyedStringDataWriter::register_instance_w_timestamp (
            const DDS_KeyedString & instance_data,
            const DDS_Time_t & source_timestamp ) [virtual]
```

Performs the same functions as **DDSKeyedStringDataWriter::register_instance** (p. 1781) except that the application provides the value for the `source_timestamp`.

See also

FooDataWriter::register_instance_w_timestamp (p. 1947)

9.266.2.5 register_instance_w_timestamp() [2/2]

```
virtual  DDS_InstanceHandle_t DDSKeyedStringDataWriter::register_instance_w_timestamp (
            const char * key,
            const  DDS_Time_t & source_timestamp )  [virtual]
```

<<***extension***>> *(p. 432)* Performs the same functions as **DDSKeyedStringDataWriter::register_instance** (p. 1781) except that the application provides the value for the `source_timestamp`.

See also

FooDataWriter::register_instance_w_timestamp (p. 1947)

9.266.2.6 unregister_instance() [1/2]

```
virtual  DDS_ReturnCode_t DDSKeyedStringDataWriter::unregister_instance (
            const  DDS_KeyedString & instance_data,
            const  DDS_InstanceHandle_t & handle )  [virtual]
```

Reverses the action of **DDSKeyedStringDataWriter::register_instance** (p. 1781).

See also

FooDataWriter::unregister_instance (p. 1948)

9.266.2.7 unregister_instance() [2/2]

```
virtual  DDS_ReturnCode_t DDSKeyedStringDataWriter::unregister_instance (
            const char * key,
            const  DDS_InstanceHandle_t & handle )  [virtual]
```

<<***extension***>> *(p. 432)* Reverses the action of **DDSKeyedStringDataWriter::register_instance** (p. 1781).

See also

FooDataWriter::unregister_instance (p. 1948)

9.266.2.8 unregister_instance_w_timestamp() [1/2]

```
virtual DDS_ReturnCode_t DDSKeyedStringDataWriter::unregister_instance_w_timestamp (
            const DDS_KeyedString & instance_data,
            const DDS_InstanceHandle_t & handle,
            const DDS_Time_t & source_timestamp ) [virtual]
```

Performs the same function as **DDSKeyedStringDataWriter::unregister_instance** (p. 1782) except that it also provides the value for the `source_timestamp`.

See also

FooDataWriter::unregister_instance_w_timestamp (p. 1950)

9.266.2.9 unregister_instance_w_timestamp() [2/2]

```
virtual DDS_ReturnCode_t DDSKeyedStringDataWriter::unregister_instance_w_timestamp (
            const char * key,
            const DDS_InstanceHandle_t & handle,
            const DDS_Time_t & source_timestamp ) [virtual]
```

<<***extension***>> *(p. 432)* Performs the same function as **DDSKeyedStringDataWriter::unregister_instance** (p. 1782) except that it also provides the value for the `source_timestamp`.

See also

FooDataWriter::unregister_instance_w_timestamp (p. 1950)

9.266.2.10 create_data()

```
virtual DDS_KeyedString * DDSKeyedStringDataWriter::create_data (
            const DDS_TypeAllocationParams_t & alloc_params ) [virtual]
```

Creates a keyed string data instance.

The size of the instance including the NULL terminated character is determined by the DataWriter property **dds.builtin_type.keyed_string.alloc_size**.

Default size: **dds.builtin_type.keyed_string.max_size** property of DomainParticipant if defined. Otherwise 1024.

Created instances must be deleted with **DDSKeyedStringDataWriter::delete_data** (p. 1783).

Returns

Newly created keyed string data, or NULL on failure.

See also

BuiltinTypeMemoryManagement section of Built-in Types

9.266.2.11 delete_data()

```
virtual  DDS_Boolean DDSKeyedStringDataWriter::delete_data (
             DDS_KeyedString * sample,
            const  DDS_TypeDeallocationParams_t & dealloc_params )  [virtual]
```

Destroys a keyed string data instance created by **DDSKeyedStringDataWriter::create_data** (p. 1783).

Returns

DDS_BOOLEAN_TRUE (p. 513) upon successful deletion.

9.266.2.12 write() [1/2]

```
virtual  DDS_ReturnCode_t DDSKeyedStringDataWriter::write (
            const  DDS_KeyedString & instance_data,
            const  DDS_InstanceHandle_t & handle )  [virtual]
```

Modifies the value of a **DDS_KeyedString** (p. 1171) data instance.

See also

FooDataWriter::write (p. 1951)

9.266.2.13 write() [2/2]

```
virtual  DDS_ReturnCode_t DDSKeyedStringDataWriter::write (
            const char * key,
            const char * str,
            const  DDS_InstanceHandle_t & handle )  [virtual]
```

<<***extension***>> *(p. 432)* Modifies the value of a **DDS_KeyedString** (p. 1171) data instance.

See also

FooDataWriter::write (p. 1951)

9.266.2.14 write_w_timestamp() [1/2]

```
virtual  DDS_ReturnCode_t DDSKeyedStringDataWriter::write_w_timestamp (
             const  DDS_KeyedString & instance_data,
             const  DDS_InstanceHandle_t & handle,
             const  DDS_Time_t & source_timestamp )  [virtual]
```

Performs the same function as **DDSKeyedStringDataWriter::write** (p. 1784) except that it also provides the value for the `source_timestamp`.

See also

FooDataWriter::write_w_timestamp (p. 1955)

9.266.2.15 write_w_timestamp() [2/2]

```
virtual  DDS_ReturnCode_t DDSKeyedStringDataWriter::write_w_timestamp (
             const char * key,
             const char * str,
             const  DDS_InstanceHandle_t & handle,
             const  DDS_Time_t & source_timestamp )  [virtual]
```

<<***extension***>> *(p. 432)* Performs the same function as **DDSKeyedStringDataWriter::write** (p. 1784) except that it also provides the value for the `source_timestamp`.

See also

FooDataWriter::write_w_timestamp (p. 1955)

9.266.2.16 write_w_params() [1/2]

```
virtual  DDS_ReturnCode_t DDSKeyedStringDataWriter::write_w_params (
             const  DDS_KeyedString & instance_data,
              DDS_WriteParams_t & params )  [virtual]
```

Performs the same function as **DDSKeyedStringDataWriter::write** (p. 1784) except that it also allows specification of the instance handle, source timestamp, publication priority, and cookie.

See also

FooDataWriter::write_w_params (p. 1956)

9.266.2.17 write_w_params() [2/2]

```
virtual DDS_ReturnCode_t DDSKeyedStringDataWriter::write_w_params (
            const char * key,
            const char * str,
             DDS_WriteParams_t & params ) [virtual]
```

<<***extension***>> *(*p. *432)* Performs the same function as **DDSKeyedStringDataWriter::write** (p. 1784) except that it also allows specification of the instance handle, source timestamp, publication priority, and cookie.

See also

FooDataWriter::write_w_params (p. 1956)

9.266.2.18 dispose() [1/2]

```
virtual DDS_ReturnCode_t DDSKeyedStringDataWriter::dispose (
            const DDS_KeyedString & instance_data,
            const DDS_InstanceHandle_t & handle ) [virtual]
```

Requests the middleware to delete the data.

See also

FooDataWriter::dispose (p. 1957)

9.266.2.19 dispose() [2/2]

```
virtual DDS_ReturnCode_t DDSKeyedStringDataWriter::dispose (
            const char * key,
            const DDS_InstanceHandle_t & instance_handle ) [virtual]
```

<<***extension***>> *(*p. *432)* Requests the middleware to delete the data.

See also

FooDataWriter::dispose (p. 1957)

9.266.2.20 dispose_w_timestamp() [1/2]

```
virtual  DDS_ReturnCode_t DDSKeyedStringDataWriter::dispose_w_timestamp (
             const  DDS_KeyedString & instance_data,
             const  DDS_InstanceHandle_t & handle,
             const  DDS_Time_t & source_timestamp )  [virtual]
```

Performs the same functions as **DDSKeyedStringDataWriter::dispose** (p. 1786) except that the application provides the value for the `source_timestamp` that is made available to **DDSDataReader** (p. 1556) objects by means of the `source_timestamp` attribute inside the **DDS_SampleInfo** (p. 1344).

See also

FooDataWriter::dispose_w_timestamp (p. 1958)

9.266.2.21 dispose_w_timestamp() [2/2]

```
virtual  DDS_ReturnCode_t DDSKeyedStringDataWriter::dispose_w_timestamp (
             const char * key,
             const  DDS_InstanceHandle_t & instance_handle,
             const  DDS_Time_t & source_timestamp )  [virtual]
```

<<***extension***>> *(p. 432)* Performs the same functions as **DDSKeyedStringDataWriter::dispose** (p. 1786) except that the application provides the value for the `source_timestamp` that is made available to **DDSDataReader** (p. 1556) objects by means of the `source_timestamp` attribute inside the **DDS_SampleInfo** (p. 1344).

See also

FooDataWriter::dispose_w_timestamp (p. 1958)

9.266.2.22 get_key_value() [1/2]

```
virtual  DDS_ReturnCode_t DDSKeyedStringDataWriter::get_key_value (
              DDS_KeyedString & key_holder,
             const  DDS_InstanceHandle_t & handle )  [virtual]
```

Retrieve the instance `key` that corresponds to an instance `handle`.

See also

FooDataWriter::get_key_value (p. 1959)

9.266.2.23 get_key_value() [2/2]

```
virtual  DDS_ReturnCode_t DDSKeyedStringDataWriter::get_key_value (
            char * key,
            const  DDS_InstanceHandle_t & handle )  [virtual]
```

<<***extension***>> *(p. 432)* Retrieve the instance `key` that corresponds to an instance `handle`.

See also

FooDataWriter::get_key_value (p. 1959)

9.266.2.24 lookup_instance() [1/2]

```
virtual  DDS_InstanceHandle_t DDSKeyedStringDataWriter::lookup_instance (
            const  DDS_KeyedString & key_holder )  [virtual]
```

Retrieve the instance `handle` that corresponds to an instance `key_holder`.

See also

FooDataWriter::lookup_instance (p. 1960)

9.266.2.25 lookup_instance() [2/2]

```
virtual  DDS_InstanceHandle_t DDSKeyedStringDataWriter::lookup_instance (
            const char * key )  [virtual]
```

<<***extension***>> *(p. 432)* Retrieve the instance `handle` that corresponds to an instance `key`.

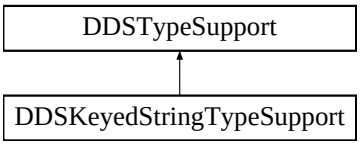
See also

FooDataWriter::lookup_instance (p. 1960)

9.267 DDSKeyedStringTypeSupport Class Reference

<<***interface***>> *(p. 433)* Keyed string type support.

Inheritance diagram for DDSKeyedStringTypeSupport:



Static Public Member Functions

- static **DDS_ReturnCode_t register_type** (**DDSDomainParticipant** *participant, const char *type_name="DDS::KeyedString")

  *Allows an application to communicate to RTI Connext the existence of the **DDS_KeyedString*** (p. *1171) data type.*
- static **DDS_ReturnCode_t unregister_type** (**DDSDomainParticipant** *participant, const char *type_name="DDS::KeyedString")

  *Allows an application to unregister the **DDS_KeyedString*** (p. *1171) data type from RTI Connext. After calling unregister_type, no further communication using this type is possible.*
- static const char * **get_type_name** ()

  *Get the default name for the **DDS_KeyedString*** (p. *1171) type.*
- static void **print_data** (const **DDS_KeyedString** *a_data)

  *<<**extension**>> (p. 432) Print value of data type to standard out.*
- static **DDS_TypeCode** * **get_typecode** ()

  *<<**extension**>> (p. 432) Retrieves the TypeCode for the Type.*
- static **DDS_ReturnCode_t serialize_data_to_cdr_buffer** (char *buffer, unsigned int &length, const **DDS_KeyedString** *a_data)

  *<<**extension**>> (p. 432) Serializes the input sample into a CDR buffer of octets.*
- static **DDS_ReturnCode_t serialize_data_to_cdr_buffer_ex** (char *buffer, unsigned int &length, const **DDS_KeyedString** *a_data, **DDS_DataRepresentationId_t** representation)

  *<<**extension**>> (p. 432) Serializes the input sample into a buffer of octets.*
- static **DDS_ReturnCode_t deserialize_data_from_cdr_buffer** (**DDS_KeyedString** *a_data, const char *buffer, unsigned int length)

  *<<**extension**>> (p. 432) Deserializes a sample from a buffer of octets.*
- static **DDS_ReturnCode_t data_to_string** (**DDS_KeyedString** *sample, char *str, **DDS_UnsignedLong** &str_size, const **DDS_PrintFormatProperty** &property)

  *<<**extension**>> (p. 432) Get the string representation of an input sample.*

9.267.1 Detailed Description

*<<**interface**>> (p. 433)* Keyed string type support.

9.267.2 Member Function Documentation

9.267.2.1 register_type()

```
static DDS_ReturnCode_t DDSKeyedStringTypeSupport::register_type (
            DDSDomainParticipant * participant,
          const char * type_name = "DDS::KeyedString" )  [static]
```

Allows an application to communicate to RTI Connext the existence of the **DDS_KeyedString** (p. 1171) data type.

By default, The **DDS_KeyedString** (p. 1171) built-in type is automatically registered when a DomainParticipant is created using the type_name returned by **DDSKeyedStringTypeSupport::get_type_name** (p. 1792). Therefore, the usage of this function is optional and it is only required when the automatic built-in type registration is disabled using the participant property "dds.builtin_type.auto_register".

This method can also be used to register the same **DDSKeyedStringTypeSupport** (p. 1788) with a **DDSDomainParticipant** (p. 1619) using different values for the type_name.

If `register_type` is called multiple times with the same **DDSDomainParticipant** (p. 1619) and `type_name`, the second (and subsequent) registrations are ignored by the operation.

Parameters

participant	<<***in***>> *(*p. *433)* the **DDSDomainParticipant** (p. 1619) to register the data type **DDS_KeyedString** (p. 1171) with. Cannot be NULL.
type_name	<<***in***>> *(*p. *433)* the type name under with the data type **DDS_KeyedString** (p. 1171) is registered with the participant; this type name is used when creating a new **DDSTopic** (p. 1885). (See **DDSDomainParticipant::create_topic** (p. 1650).) The name may not be NULL or longer than 255 characters.

Exceptions

One	of the **Standard Return Codes** (p. 532), **DDS_RETCODE_PRECONDITION_NOT_MET** (p. 532) or **DDS_RETCODE_OUT_OF_RESOURCES** (p. 533).

MT Safety:

UNSAFE on the FIRST call. It is not safe for two threads to simultaneously make the first call to register a type. Subsequent calls are thread safe.

See also

DDSDomainParticipant::create_topic (p. 1650)

9.267.2.2 unregister_type()

```
static  DDS_ReturnCode_t DDSKeyedStringTypeSupport::unregister_type (
              DDSDomainParticipant * participant,
            const char * type_name = "DDS::KeyedString" )  [static]
```

Allows an application to unregister the **DDS_KeyedString** (p. 1171) data type from RTI Connext. After calling unregister_type, no further communication using this type is possible.

Precondition

The **DDS_KeyedString** (p. 1171) type with `type_name` is registered with the participant and all **DDSTopic** (p. 1885) objects referencing the type have been destroyed. If the type is not registered with the participant, or if any **DDSTopic** (p. 1885) is associated with the type, the operation will fail with **DDS_RETCODE_ERROR** (p. 532).

Postcondition

All information about the type is removed from RTI Connext. No further communication using this type is possible.

Parameters

participant	<<***in***>> *(*p. *433)* the **DDSDomainParticipant** (p. 1619) to unregister the data type **DDS_KeyedString** (p. 1171) from. Cannot be NULL.
type_name	<<***in***>> *(*p. *433)* the type name under with the data type **DDS_KeyedString** (p. 1171) is registered with the participant. The name should match a name that has been previously used to register a type with the participant. Cannot be NULL.

Exceptions

One	of the **Standard Return Codes** (p. 532), **DDS_RETCODE_BAD_PARAMETER** (p. 532) or **DDS_RETCODE_ERROR** (p. 532)

MT Safety:

SAFE.

See also

DDSKeyedStringTypeSupport::register_type (p. 1789)

9.267.2.3 get_type_name()

```
static const char * DDSKeyedStringTypeSupport::get_type_name ( )  [static]
```

Get the default name for the **DDS_KeyedString** (p. 1171) type.

Can be used for calling **DDSKeyedStringTypeSupport::register_type** (p. 1789) or creating **DDSTopic** (p. 1885).

Returns

default name for the **DDS_KeyedString** (p. 1171) type.

See also

DDSKeyedStringTypeSupport::register_type (p. 1789)

DDSDomainParticipant::create_topic (p. 1650)

9.267.2.4 print_data()

```
static void DDSKeyedStringTypeSupport::print_data (
            const  DDS_KeyedString * a_data )  [static]
```

<<***extension***>> *(*p. *432)* Print value of data type to standard out.

The *generated* implementation of the operation knows how to print value of a data type.

Parameters

a_data	<<***in***>> *(p. 433)* **DDS_KeyedString** (p. 1171) to be printed.

9.267.2.5 get_typecode()

```
static  DDS_TypeCode * DDSKeyedStringTypeSupport::get_typecode ( )  [static]
```

<<***extension***>> *(p. 432)* Retrieves the TypeCode for the Type.

See also

FooTypeSupport::get_typecode (p. 1991)

9.267.2.6 serialize_data_to_cdr_buffer()

```
static  DDS_ReturnCode_t DDSKeyedStringTypeSupport::serialize_data_to_cdr_buffer (
             char * buffer,
             unsigned int & length,
             const  DDS_KeyedString * a_data )  [static]
```

<<***extension***>> *(p. 432)* Serializes the input sample into a CDR buffer of octets.

See also

FooTypeSupport::serialize_data_to_cdr_buffer (p. 1989)

9.267.2.7 serialize_data_to_cdr_buffer_ex()

```
static  DDS_ReturnCode_t DDSKeyedStringTypeSupport::serialize_data_to_cdr_buffer_ex (
             char * buffer,
             unsigned int & length,
             const  DDS_KeyedString * a_data,
              DDS_DataRepresentationId_t representation )  [static]
```

<<***extension***>> *(p. 432)* Serializes the input sample into a buffer of octets.

See also

FooTypeSupport::serialize_data_to_cdr_buffer (p. 1989)

9.267.2.8 deserialize_data_from_cdr_buffer()

```
static DDS_ReturnCode_t DDSKeyedStringTypeSupport::deserialize_data_from_cdr_buffer (
            DDS_KeyedString * a_data,
            const char * buffer,
            unsigned int length )  [static]
```

<<***extension***>> *(p. 432)* Deserializes a sample from a buffer of octets.

See also

FooTypeSupport::deserialize_data_from_cdr_buffer (p. 1990)

9.267.2.9 data_to_string()

```
static DDS_ReturnCode_t DDSKeyedStringTypeSupport::data_to_string (
            DDS_KeyedString * sample,
            char * str,
            DDS_UnsignedLong & str_size,
            const DDS_PrintFormatProperty & property )  [static]
```

<<***extension***>> *(p. 432)* Get the string representation of an input sample.

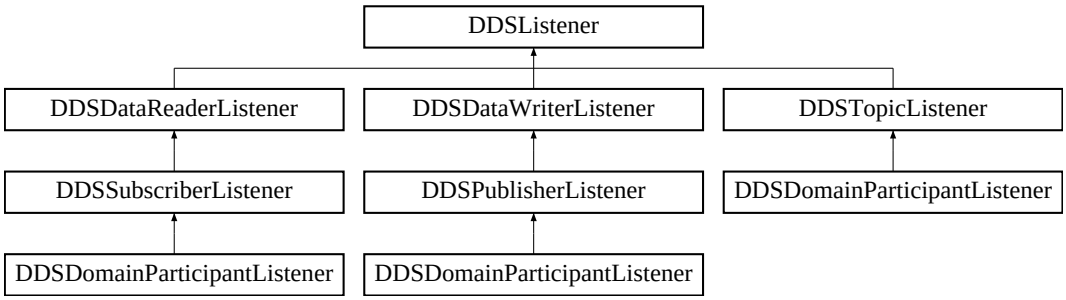
See also

FooTypeSupport::data_to_string (p. 1991)

9.268 DDSListener Class Reference

<<***interface***>> *(p. 433)* Abstract base class for all Listener interfaces.

Inheritance diagram for DDSListener:



9.268.1 Detailed Description

<<***interface***>> *(*p. *433)* Abstract base class for all Listener interfaces.

Entity:

DDSEntity (p. 1731)

QoS:

All QoS Policies (p. 549)

Status:

Status Kinds (p. 533)

All the supported kinds of concrete **DDSListener** (p. 1794) interfaces (one per concrete **DDSEntity** (p. 1731) type) derive from this root and add methods whose prototype depends on the concrete Listener.

Listeners provide a way for RTI Connext to asynchronously alert the application when there are relevant status changes.

Almost every application will have to implement listener interfaces.

Each dedicated listener presents a list of operations that correspond to the relevant communication status changes to which an application may respond.

The same **DDSListener** (p. 1794) instance may be shared among multiple entities if you so desire. Consequently, the provided parameter contains a reference to the concerned **DDSEntity** (p. 1731).

9.268.2 Access to Plain Communication Status

The general mapping between the plain communication statuses (see **Status Kinds** (p. 533)) and the listeners' operations is as follows:

- For each communication status, there is a corresponding operation whose name is `on_<communication_status>`(), which takes a parameter of type `<communication_status>` as listed in **Status Kinds** (p. 533).
- `on_<communication_status>` is available on the relevant **DDSEntity** (p. 1731) as well as those that embed it, as expressed in the following figure:

listener processing. The most *specific* relevant enabled listener is called."

- When the application attaches a listener on an entity, it must set a mask. The mask indicates to RTI Connext which operations are enabled within the listener (cf. operation **DDSEntity** (p. 1731) set_listener()).
- When a plain communication status changes, RTI Connext triggers the most specific relevant listener operation that is enabled. In case the most specific relevant listener operation corresponds to an application-installed 'nil' listener the operation will be considered handled by a NO-OP operation that does not reset the communication status.

This behavior allows the application to set a default behavior (e.g., in the listener associated with the **DDSDomain↩ Participant** (p. 1619)) and to set dedicated behaviors only where needed.

9.268.3 Access to Read Communication Status

The two statuses related to data arrival are treated slightly differently. Since they constitute the core purpose of the Data Distribution Service, there is no need to provide a default mechanism (as is done for the plain communication statuses above).

The rule is as follows. Each time the read communication status changes:

- First, RTI Connext tries to trigger the **DDSSubscriberListener::on_data_on_readers** (p. 1882) with a parameter of the related **DDSSubscriber** (p. 1860);
- If this does not succeed (there is no listener or the operation is not enabled), RTI Connext tries to trigger **DDSDataReaderListener::on_data_available** (p. 1585) on all the related **DDSDataReaderListener** (p. 1583) objects, with a parameter of the related **DDSDataReader** (p. 1556).

The rationale is that either the application is interested in relations among data arrivals and it must use the first option (and then get the corresponding **DDSDataReader** (p. 1556) objects by calling **DDSSubscriber::get_datareaders** (p. 1874) on the related **DDSSubscriber** (p. 1860) and then get the data by calling **FooDataReader::read** (p. 1919) or **FooDataReader::take** (p. 1920) on the returned **DDSDataReader** (p. 1556) objects), or it wants to treat each **DDSDataReader** (p. 1556) independently and it may choose the second option (and then get the data by calling **Foo↩ DataReader::read** (p. 1919) or **FooDataReader::take** (p. 1920) on the related **DDSDataReader** (p. 1556)).

Note that if **DDSSubscriberListener::on_data_on_readers** (p. 1882) is called, RTI Connext will *not* try to call **DDSDataReaderListener::on_data_available** (p. 1585). However, an application can force a call to the **DDSData↩ Reader** (p. 1556) objects that have data by calling **DDSSubscriber::notify_datareaders** (p. 1876).

9.268.4 Operations Allowed in Listener Callbacks

See `Restricted Operations in Listener Callbacks`, in the Core Libraries User's Manual.

9.268.5 Best Practices with Listeners

Note that all the issues described below are avoided by using **DDSWaitSet** (p. 1897).

Avoid blocking or performing a lot of processing in Listener callbacks

Listeners are invoked by internal threads that perform critical functions within the middleware and need to run in a timely manner. By default, Connext DDS creates a few threads to use to receive data and only a single thread to handle periodic events.

Because of this, user applications installing Listeners should never block in a Listener callback. There are several negative consequences of blocking in a listener callback:

- The application may lose data for the DataReader the listener is installed on, because the receive thread is not removing it from the socket buffer and it gets overwritten.

- The application may receive strictly reliable data with a delay, because the receive thread is not removing it from the socket buffer and if it gets overwritten it must be re-sent.
- The application may lose or delay data for other DataReaders, because by default all DataReaders created with the same DomainParticipant share the same threads.
- The application may not be notified of periodic events on time

If the application needs to make a blocking call when data is available, or when another event occurs, the application should use **DDSWaitSet** (p. 1897).

Avoid taking application mutexes/semaphores in Listener callbacks

Taking application mutexes/sempahores within a Listener callback may lead to unexpected deadlock scenarios.

When a Listener callback is invoked the EA (Exclusive Area) of the Entity 'E' to which the callback applies is taken by the middleware.

If the application takes an application mutex 'M' within a critical section in which the application makes DDS calls affecting 'E', this may lead to following deadlock:

The middleware thread is within the entity EA trying to acquire the mutex 'M'. At the same time, the application thread has acquired 'M' and is blocked trying to acquire the entity EA.

Do not write data with a DataWriter within the on_data_available callback

Avoid writing data with a DataWriter within the **DDSDataReaderListener::on_data_available()** (p. 1585) callback. If the write operation blocks because e.g. the send window is full, this will lead to a deadlock.

Do not call wait_for_acknowledgements within the on_data_available callback

Do not call the **DDSDataWriter::wait_for_acknowledgments** (p. 1602) within the **DDSDataReaderListener::on_↩ data_available()** (p. 1585) callback. This will lead to deadlock.

See also

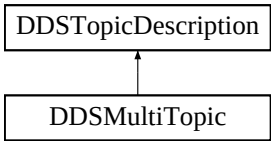
Status Kinds (p. 533)

DDSWaitSet (p. 1897), **DDSCondition** (p. 1544)

9.269 DDSMultiTopic Class Reference

[Not supported (optional)] <<***interface***>> *(p. 433)* A specialization of **DDSTopicDescription** (p. 1892) that allows subscriptions that combine/filter/rearrange data coming from several topics.

Inheritance diagram for DDSMultiTopic:



Public Member Functions

- virtual const char * **get_subscription_expression** ()=0

  *Get the expression for this **DDSMultiTopic** (p. 1797).*
- virtual **DDS_ReturnCode_t** **get_expression_parameters** (**DDS_StringSeq** &parameters)=0

  Get the expression parameters.
- virtual **DDS_ReturnCode_t** **set_expression_parameters** (const **DDS_StringSeq** &parameters)=0

  Set the `expression_parameters`*.*

Static Public Member Functions

- static **DDSMultiTopic** * **narrow** (**DDSTopicDescription** *topic_description)

  *Narrow the given **DDSTopicDescription** (p. 1892) pointer to a **DDSMultiTopic** (p. 1797) pointer.*

9.269.1 Detailed Description

[Not supported (optional)] <<***interface***>> *(p. 433)* A specialization of **DDSTopicDescription** (p. 1892) that allows subscriptions that combine/filter/rearrange data coming from several topics.

DDSMultiTopic (p. 1797) allows a more sophisticated subscription that can select and combine data received from multiple topics into a single resulting type (specified by the inherited `type_name`). The data will then be filtered (selection) and possibly re-arranged (aggregation/projection) according to a `subscription_expression` with parameters `expression_parameters`.

- The `subscription_expression` is a string that identifies the selection and re-arrangement of data from the associated topics. It is similar to an SQL statement where the SELECT part provides the fields to be kept, the FROM part provides the names of the topics that are searched for those fields, and the WHERE clause gives the content filter. The Topics combined may have different types but they are restricted in that the type of the fields used for the NATURAL JOIN operation must be the same.

- The `expression_parameters` attribute is a sequence of strings that give values to the 'parameters' (i.↩ e. "%n" tokens) in the `subscription_expression`. The number of supplied parameters must fit with the requested values in the `subscription_expression` (i.e. the number of n tokens).

- **DDSDataReader** (p. 1556) entities associated with a **DDSMultiTopic** (p. 1797) are alerted of data modifications by the usual **DDSListener** (p. 1794) or **DDSWaitSet** (p. 1897) / **DDSCondition** (p. 1544) mechanisms whenever modifications occur to the data associated with any of the topics relevant to the **DDSMultiTopic** (p. 1797).

Note that the source for data may not be restricted to a single topic.

DDSDataReader (p. 1556) entities associated with a **DDSMultiTopic** (p. 1797) may access instances that are "constructed" at the **DDSDataReader** (p. 1556) side from the instances written by multiple **DDSDataWriter** (p. 1589) entities. The **DDSMultiTopic** (p. 1797) access instance will begin to exist as soon as all the constituting **DDSTopic** (p. 1885) instances are in existence. The `view_state` and `instance_state` is computed from the corresponding states of the constituting instances:

- The view_state of the **DDSMultiTopic** (p. 1797) instance is **DDS_NEW_VIEW_STATE** (p. 164) if at least one of the constituting instances has view_state = **DDS_NEW_VIEW_STATE** (p. 164). Otherwise, it will be **DDS_NOT_NEW_VIEW_STATE** (p. 164).
- The instance_state of the **DDSMultiTopic** (p. 1797) instance is **DDS_ALIVE_INSTANCE_STATE** (p. 166) if the instance_state of all the constituting **DDSTopic** (p. 1885) instances is **DDS_ALIVE_INSTANCE_STATE** (p. 166). It is **DDS_NOT_ALIVE_DISPOSED_INSTANCE_STATE** (p. 166) if at least one of the constituting **DDSTopic** (p. 1885) instances is **DDS_NOT_ALIVE_DISPOSED_INSTANCE_STATE** (p. 166). Otherwise, it is **DDS_NOT_ALIVE_NO_WRITERS_INSTANCE_STATE** (p. 166).

Queries and Filters Syntax (p. 184) describes the syntax of subscription_expression and expression_parameters.

9.269.2 Member Function Documentation

9.269.2.1 narrow()

```
static DDSMultiTopic * DDSMultiTopic::narrow (
            DDSTopicDescription * topic_description ) [static]
```

Narrow the given **DDSTopicDescription** (p. 1892) pointer to a **DDSMultiTopic** (p. 1797) pointer.

Returns

DDSMultiTopic (p. 1797) if this **DDSTopicDescription** (p. 1892) is a **DDSMultiTopic** (p. 1797). Otherwise, return NULL.

9.269.2.2 get_subscription_expression()

```
virtual const char * DDSMultiTopic::get_subscription_expression ( ) [pure virtual]
```

Get the expression for this **DDSMultiTopic** (p. 1797).

The expressions syntax is described in the DDS specification. It is specified when the **DDSMultiTopic** (p. 1797) is created.

Returns

subscription_expression of the **DDSMultiTopic** (p. 1797).

9.269.2.3 get_expression_parameters()

```
virtual DDS_ReturnCode_t DDSMultiTopic::get_expression_parameters (
            DDS_StringSeq & parameters ) [pure virtual]
```

Get the expression parameters.

The expressions syntax is described in the DDS specification.

The parameters is either specified on the last successful call to **DDSMultiTopic::set_expression_parameters** (p. 1800), or if **DDSMultiTopic::set_expression_parameters** (p. 1800) was never called, the parameters specified when the **DDSMultiTopic** (p. 1797) was created.

Parameters

parameters	<<***inout***>> *(p. 433)* Fill in this sequence with the expression parameters. The memory for the strings in this sequence is managed according to the conventions described in **String Conventions** (p. 758). In particular, be careful to avoid a situation in which RTI Connext allocates a string on your behalf and you then reuse that string in such a way that RTI Connext believes it to have more memory allocated to it than it actually does.

Exceptions

One	of the **Standard Return Codes** (p. 532)

9.269.2.4 set_expression_parameters()

```
virtual  DDS_ReturnCode_t DDSMultiTopic::set_expression_parameters (
             const  DDS_StringSeq & parameters )  [pure virtual]
```

Set the `expression_parameters`.

Changes the `expression_parameters` associated with the **DDSMultiTopic** (p. 1797).

Parameters

parameters	<<***in***>> *(p. 433)* the filter expression parameters

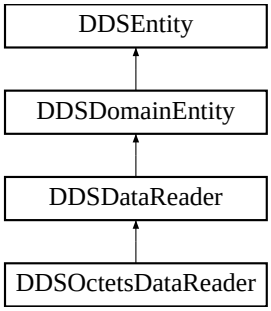
Returns

One of the **Standard Return Codes** (p. 532).

9.270 DDSOctetsDataReader Class Reference

<<***interface***>> *(p. 433)* Instantiates `DataReader` < **DDS_Octets** (p. 1217) >.

Inheritance diagram for DDSOctetsDataReader:



Public Member Functions

- virtual **DDS_ReturnCode_t read** (**DDS_OctetsSeq** &received_data, **DDS_SampleInfoSeq** &info_seq, **DDS_Long** max_samples= **DDS_LENGTH_UNLIMITED**, **DDS_SampleStateMask** sample_states= **DDS_ANY_SAMPLE_STATE**, **DDS_ViewStateMask** view_states= **DDS_ANY_VIEW_STATE**, **DDS_InstanceStateMask** instance_states= **DDS_ANY_INSTANCE_STATE**)

  *Access a collection of data samples from the **DDSDataReader** (p. 1556).*
- virtual **DDS_ReturnCode_t take** (**DDS_OctetsSeq** &received_data, **DDS_SampleInfoSeq** &info_seq, **DDS_Long** max_samples= **DDS_LENGTH_UNLIMITED**, **DDS_SampleStateMask** sample_states= **DDS_ANY_SAMPLE_STATE**, **DDS_ViewStateMask** view_states= **DDS_ANY_VIEW_STATE**, **DDS_InstanceStateMask** instance_states= **DDS_ANY_INSTANCE_STATE**)

  *Access a collection of data-samples from the **DDSDataReader** (p. 1556).*
- virtual **DDS_ReturnCode_t read_w_condition** (**DDS_OctetsSeq** &received_data, **DDS_SampleInfoSeq** &info_seq, **DDS_Long** max_samples, **DDSReadCondition** *condition)

  *Accesses via **DDSOctetsDataReader::read** (p. 1802) the samples that match the criteria specified in the **DDSReadCondition** (p. 1842).*
- virtual **DDS_ReturnCode_t take_w_condition** (**DDS_OctetsSeq** &received_data, **DDS_SampleInfoSeq** &info_seq, **DDS_Long** max_samples, **DDSReadCondition** *condition)

  *Analogous to **DDSOctetsDataReader::read_w_condition** (p. 1802) except it accesses samples via the **DDSOctetsDataReader::take** (p. 1802) operation.*
- virtual **DDS_ReturnCode_t read_next_sample** (**DDS_Octets** &received_data, **DDS_SampleInfo** &sample_info)

  *Copies the next not-previously-accessed data value from the **DDSDataReader** (p. 1556).*
- virtual **DDS_ReturnCode_t take_next_sample** (**DDS_Octets** &received_data, **DDS_SampleInfo** &sample_info)

  *Copies the next not-previously-accessed data value from the **DDSDataReader** (p. 1556).*
- virtual **DDS_ReturnCode_t return_loan** (**DDS_OctetsSeq** &received_data, **DDS_SampleInfoSeq** &info_seq)

  *Indicates to the **DDSDataReader** (p. 1556) that the application is done accessing the collection of `received_data` and `info_seq` obtained by some earlier invocation of read or take on the **DDSDataReader** (p. 1556).*

Static Public Member Functions

- static **DDSOctetsDataReader** * **narrow** (**DDSDataReader** *reader)

  *Narrow the given **DDSDataReader** (p. 1556) pointer to a **DDSOctetsDataReader** (p. 1800) pointer.*

9.270.1 Detailed Description

<<***interface***>> *(p. 433)* Instantiates `DataReader` < **DDS_Octets** (p. 1217) >.

When reading or taking data with this reader, if you request a copy of the samples instead of a loan, and the byte array in a destination data sample is NULL, the middleware will allocate a new array for you of sufficient length to hold the received data. The new array will be allocated with **DDS_OctetBuffer_alloc** (p. 755); the sample's destructor will delete it.

A non- NULL array is assumed to be allocated to sufficient length to store the incoming data. It will not be reallocated.

See also

FooDataReader (p. 1917)

DDSDataReader (p. 1556)

9.270.2 Member Function Documentation

9.270.2.1 read()

```
virtual DDS_ReturnCode_t DDSOctetsDataReader::read (
            DDS_OctetsSeq & received_data,
            DDS_SampleInfoSeq & info_seq,
            DDS_Long max_samples = DDS_LENGTH_UNLIMITED,
            DDS_SampleStateMask sample_states = DDS_ANY_SAMPLE_STATE,
            DDS_ViewStateMask view_states = DDS_ANY_VIEW_STATE,
            DDS_InstanceStateMask instance_states = DDS_ANY_INSTANCE_STATE ) [virtual]
```

Access a collection of data samples from the **DDSDataReader** (p. 1556).

See also

FooDataReader::read (p. 1919)

9.270.2.2 take()

```
virtual DDS_ReturnCode_t DDSOctetsDataReader::take (
            DDS_OctetsSeq & received_data,
            DDS_SampleInfoSeq & info_seq,
            DDS_Long max_samples = DDS_LENGTH_UNLIMITED,
            DDS_SampleStateMask sample_states = DDS_ANY_SAMPLE_STATE,
            DDS_ViewStateMask view_states = DDS_ANY_VIEW_STATE,
            DDS_InstanceStateMask instance_states = DDS_ANY_INSTANCE_STATE ) [virtual]
```

Access a collection of data-samples from the **DDSDataReader** (p. 1556).

See also

FooDataReader::take (p. 1920)

9.270.2.3 read_w_condition()

```
virtual DDS_ReturnCode_t DDSOctetsDataReader::read_w_condition (
            DDS_OctetsSeq & received_data,
            DDS_SampleInfoSeq & info_seq,
            DDS_Long max_samples,
            DDSReadCondition * condition ) [virtual]
```

Accesses via **DDSOctetsDataReader::read** (p. 1802) the samples that match the criteria specified in the **DDSReadCondition** (p. 1842).

See also

FooDataReader::read_w_condition (p. 1925)

9.270.2.4 take_w_condition()

```
virtual  DDS_ReturnCode_t DDSOctetsDataReader::take_w_condition (
              DDS_OctetsSeq & received_data,
              DDS_SampleInfoSeq & info_seq,
              DDS_Long max_samples,
              DDSReadCondition * condition )  [virtual]
```

Analogous to **DDSOctetsDataReader::read_w_condition** (p. 1802) except it accesses samples via the **DDSOctets↩ DataReader::take** (p. 1802) operation.

See also

FooDataReader::take_w_condition (p. 1926)

9.270.2.5 read_next_sample()

```
virtual  DDS_ReturnCode_t DDSOctetsDataReader::read_next_sample (
              DDS_Octets & received_data,
              DDS_SampleInfo & sample_info )  [virtual]
```

Copies the next not-previously-accessed data value from the **DDSDataReader** (p. 1556).

See also

FooDataReader::read_next_sample (p. 1927)

9.270.2.6 take_next_sample()

```
virtual  DDS_ReturnCode_t DDSOctetsDataReader::take_next_sample (
              DDS_Octets & received_data,
              DDS_SampleInfo & sample_info )  [virtual]
```

Copies the next not-previously-accessed data value from the **DDSDataReader** (p. 1556).

See also

FooDataReader::take_next_sample (p. 1928)

9.270.2.7 return_loan()

```
virtual  DDS_ReturnCode_t DDSOctetsDataReader::return_loan (
              DDS_OctetsSeq & received_data,
              DDS_SampleInfoSeq & info_seq )  [virtual]
```

Indicates to the **DDSDataReader** (p. 1556) that the application is done accessing the collection of `received_data` and `info_seq` obtained by some earlier invocation of read or take on the **DDSDataReader** (p. 1556).

See also

FooDataReader::return_loan (p. 1940)

9.270.2.8 narrow()

```
static  DDSOctetsDataReader * DDSOctetsDataReader::narrow (
              DDSDataReader * reader )  [static]
```

Narrow the given **DDSDataReader** (p. 1556) pointer to a **DDSOctetsDataReader** (p. 1800) pointer.

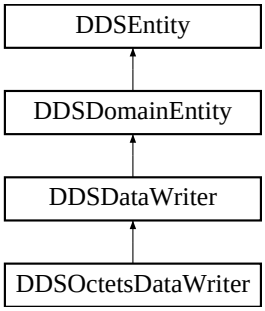
See also

FooDataReader::narrow (p. 1919)

9.271 DDSOctetsDataWriter Class Reference

<<***interface***>> *(p. 433)* Instantiates `DataWriter` < **DDS_Octets** (p. 1217) >.

Inheritance diagram for DDSOctetsDataWriter:



Public Member Functions

- virtual **DDS_Octets** * **create_data** (const **DDS_TypeAllocationParams_t** &alloc_params)

  Creates an octet data sequence.
- virtual **DDS_Boolean** **delete_data** (**DDS_Octets** *sample, const **DDS_TypeDeallocationParams_t** &dealloc_params)

  *Destroys a octet data sequence created by **DDSOctetsDataWriter::create_data** (p. 1806).*
- virtual **DDS_ReturnCode_t** **write** (const **DDS_Octets** &instance_data, const **DDS_InstanceHandle_t** &handle)

  *Modifies the value of a **DDS_Octets** (p. 1217) data instance.*
- virtual **DDS_ReturnCode_t** **write** (const unsigned char *octets, int length, const **DDS_InstanceHandle_t** &handle)

  *<<**extension**>> (p. 432) Modifies the value of a **DDS_Octets** (p. 1217) data instance.*
- virtual **DDS_ReturnCode_t** **write** (const **DDS_OctetSeq** &octets, const **DDS_InstanceHandle_t** &handle)

  *<<**extension**>> (p. 432) Modifies the value of a **DDS_Octets** (p. 1217) data instance.*
- virtual **DDS_ReturnCode_t** **write_w_timestamp** (const **DDS_Octets** &instance_data, const **DDS_InstanceHandle_t** &handle, const **DDS_Time_t** &source_timestamp)

  *Performs the same function as **DDSOctetsDataWriter::write** (p. 1807) except that it also provides the value for the `source_timestamp`.*
- virtual **DDS_ReturnCode_t** **write_w_timestamp** (const unsigned char *octets, int length, const **DDS_InstanceHandle_t** &handle, const **DDS_Time_t** &source_timestamp)

  *<<**extension**>> (p. 432) Performs the same function as **DDSOctetsDataWriter::write** (p. 1807) except that it also provides the value for the `source_timestamp`.*
- virtual **DDS_ReturnCode_t** **write_w_timestamp** (const **DDS_OctetSeq** &octets, const **DDS_InstanceHandle_t** &handle, const **DDS_Time_t** &source_timestamp)

  *<<**extension**>> (p. 432) Performs the same function as **DDSOctetsDataWriter::write** (p. 1807) except that it also provides the value for the `source_timestamp`.*
- virtual **DDS_ReturnCode_t** **write_w_params** (const **DDS_Octets** &instance_data, **DDS_WriteParams_t** &params)

  *Performs the same function as **DDSOctetsDataWriter::write** (p. 1807) except that it also allows specification of the instance handle, source timestamp, publication priority, and cookie.*
- virtual **DDS_ReturnCode_t** **write_w_params** (const unsigned char *octets, int length, **DDS_WriteParams_t** &params)

  *<<**extension**>> (p. 432) Performs the same function as **DDSOctetsDataWriter::write** (p. 1807) except that it also allows specification of the instance handle, source timestamp, publication priority, and cookie.*
- virtual **DDS_ReturnCode_t** **write_w_params** (const **DDS_OctetSeq** &octets, **DDS_WriteParams_t** &params)

  *<<**extension**>> (p. 432) Performs the same function as **DDSOctetsDataWriter::write** (p. 1807) except that it also allows specification of the instance handle, source timestamp, publication priority, and cookie.*

Static Public Member Functions

- static **DDSOctetsDataWriter** * **narrow** (**DDSDataWriter** *writer)

  *Narrow the given **DDSDataWriter** (p. 1589) pointer to a **DDSOctetsDataWriter** (p. 1804) pointer.*

9.271.1 Detailed Description

*<<**interface**>> (p. 433)* Instantiates `DataWriter` < **DDS_Octets** (p. 1217) >.

See also

FooDataWriter (p. 1944)

DDSDataWriter (p. 1589)

9.271.2 Member Function Documentation

9.271.2.1 narrow()

```
static DDSOctetsDataWriter * DDSOctetsDataWriter::narrow (
            DDSDataWriter * writer ) [static]
```

Narrow the given **DDSDataWriter** (p. 1589) pointer to a **DDSOctetsDataWriter** (p. 1804) pointer.

See also

FooDataWriter::narrow (p. 1946)

9.271.2.2 create_data()

```
virtual DDS_Octets * DDSOctetsDataWriter::create_data (
            const DDS_TypeAllocationParams_t & alloc_params ) [virtual]
```

Creates an octet data sequence.

The size of the instance is determined by the DataWriter property **dds.builtin_type.octets.alloc_size**.

Default size: **dds.builtin_type.octets.max_size** property of DomainParticipant if defined. Otherwise 2048.

Created instances must be deleted with **DDSOctetsDataWriter::delete_data** (p. 1806).

Returns

Newly created octet sequence data, or NULL on failure.

See also

BuiltinTypeMemoryManagement section of Built-in Types.

9.271.2.3 delete_data()

```
virtual  DDS_Boolean DDSOctetsDataWriter::delete_data (
              DDS_Octets * sample,
             const  DDS_TypeDeallocationParams_t & dealloc_params )  [virtual]
```

Destroys a octet data sequence created by **DDSOctetsDataWriter::create_data** (p. 1806).

Returns

DDS_BOOLEAN_TRUE (p. 513) upon successful deletion.

9.271.2.4 write() [1/3]

```
virtual  DDS_ReturnCode_t DDSOctetsDataWriter::write (
             const  DDS_Octets & instance_data,
             const  DDS_InstanceHandle_t & handle )  [virtual]
```

Modifies the value of a **DDS_Octets** (p. 1217) data instance.

See also

FooDataWriter::write (p. 1951)

9.271.2.5 write() [2/3]

```
virtual  DDS_ReturnCode_t DDSOctetsDataWriter::write (
             const unsigned char * octets,
             int length,
             const  DDS_InstanceHandle_t & handle )  [virtual]
```

<<***extension***>> *(p. 432)* Modifies the value of a **DDS_Octets** (p. 1217) data instance.

Parameters

octets	<<***in***>> *(p. 433)* Array of octets to be published.
length	<<***in***>> *(p. 433)* Number of octets to be published.
handle	<<***in***>> *(p. 433)* The special value **DDS_HANDLE_NIL** (p. 78) should be used always.

See also

FooDataWriter::write (p. 1951)

9.271.2.6 write() [3/3]

```
virtual  DDS_ReturnCode_t DDSOctetsDataWriter::write (
             const  DDS_OctetSeq & octets,
             const  DDS_InstanceHandle_t & handle )  [virtual]
```

<<***extension***>> *(p. 432)* Modifies the value of a **DDS_Octets** (p. 1217) data instance.

Parameters

octets	<<***in***>> *(p. 433)* Sequence of octets to be published.
handle	<<***in***>> *(p. 433)* The special value **DDS_HANDLE_NIL** (p. 78) should be used always.

See also

FooDataWriter::write (p. 1951)

9.271.2.7 write_w_timestamp() [1/3]

```
virtual  DDS_ReturnCode_t DDSOctetsDataWriter::write_w_timestamp (
             const  DDS_Octets & instance_data,
             const  DDS_InstanceHandle_t & handle,
             const  DDS_Time_t & source_timestamp )  [virtual]
```

Performs the same function as **DDSOctetsDataWriter::write** (p. 1807) except that it also provides the value for the `source_timestamp`.

See also

FooDataWriter::write_w_timestamp (p. 1955)

9.271.2.8 write_w_timestamp() [2/3]

```
virtual  DDS_ReturnCode_t DDSOctetsDataWriter::write_w_timestamp (
             const unsigned char * octets,
             int length,
             const  DDS_InstanceHandle_t & handle,
             const  DDS_Time_t & source_timestamp )  [virtual]
```

<<***extension***>> *(p. 432)* Performs the same function as **DDSOctetsDataWriter::write** (p. 1807) except that it also provides the value for the `source_timestamp`.

Parameters

octets	<<***in***>> *(p. 433)* Array of octets to be published.
length	<<***in***>> *(p. 433)* Number of octets to be published.
handle	<<***in***>> *(p. 433)* The special value **DDS_HANDLE_NIL** (p. 78) should be used always.
source_timestamp	<<***in***>> *(p. 433)* The timestamp value must be greater than or equal to the timestamp value used in the last writer operation. See **FooDataWriter::write_w_timestamp** (p. 1955).

See also

FooDataWriter::write_w_timestamp (p. 1955)

9.271.2.9 write_w_timestamp() [3/3]

```
virtual DDS_ReturnCode_t DDSOctetsDataWriter::write_w_timestamp (
            const DDS_OctetSeq & octets,
            const DDS_InstanceHandle_t & handle,
            const DDS_Time_t & source_timestamp ) [virtual]
```

<<***extension***>> *(p. 432)* Performs the same function as **DDSOctetsDataWriter::write** (p. 1807) except that it also provides the value for the `source_timestamp`.

Parameters

octets	<<***in***>> *(p. 433)* Sequence of octets to be published.
handle	<<***in***>> *(p. 433)* The special value **DDS_HANDLE_NIL** (p. 78) should be used always.
source_timestamp	<<***in***>> *(p. 433)* The timestamp value must be greater than or equal to the timestamp value used in the last writer operation. See **FooDataWriter::write_w_timestamp** (p. 1955).

See also

FooDataWriter::write_w_timestamp (p. 1955)

9.271.2.10 write_w_params() [1/3]

```
virtual DDS_ReturnCode_t DDSOctetsDataWriter::write_w_params (
            const DDS_Octets & instance_data,
             DDS_WriteParams_t & params ) [virtual]
```

Performs the same function as **DDSOctetsDataWriter::write** (p. 1807) except that it also allows specification of the instance handle, source timestamp, publication priority, and cookie.

See also

FooDataWriter::write_w_params (p. 1956)

9.271.2.11 write_w_params() [2/3]

```
virtual DDS_ReturnCode_t DDSOctetsDataWriter::write_w_params (
            const unsigned char * octets,
            int length,
             DDS_WriteParams_t & params ) [virtual]
```

<<***extension***>> *(p. 432)* Performs the same function as **DDSOctetsDataWriter::write** (p. 1807) except that it also allows specification of the instance handle, source timestamp, publication priority, and cookie.

Parameters

octets	<<***in***>> *(p. 433)* Array of octets to be published.
length	<<***in***>> *(p. 433)* Number of octets to be published.
params	<<***in***>> *(p. 433)* The **DDS_WriteParams_t** (p. 1519) parameter containing the instance handle, source timestamp, publication priority, and cookie to be used in write operation. See **FooDataWriter::write_w_params** (p. 1956).

See also

FooDataWriter::write_w_params (p. 1956)

9.271.2.12 write_w_params() [3/3]

```
virtual DDS_ReturnCode_t DDSOctetsDataWriter::write_w_params (
            const  DDS_OctetSeq & octets,
             DDS_WriteParams_t & params ) [virtual]
```

<<***extension***>> *(p. 432)* Performs the same function as **DDSOctetsDataWriter::write** (p. 1807) except that it also allows specification of the instance handle, source timestamp, publication priority, and cookie.

Parameters

octets	<<***in***>> *(p. 433)* Sequence of octets to be published.
params	<<***in***>> *(p. 433)* The **DDS_WriteParams_t** (p. 1519) parameter containing the instance handle, source timestamp, publication priority, and cookie to be used in write operation. See **FooDataWriter::write_w_params** (p. 1956).

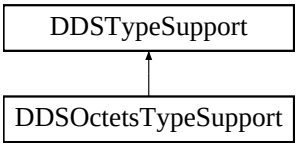
See also

FooDataWriter::write_w_params (p. 1956)

9.272 DDSOctetsTypeSupport Class Reference

<<***interface***>> *(p. 433)* **DDS_Octets** (p. 1217) type support.

Inheritance diagram for DDSOctetsTypeSupport:



Static Public Member Functions

- static **DDS_ReturnCode_t** **register_type** (**DDSDomainParticipant** *participant, const char *type_name="DDS::Octets")

  Allows an application to communicate to RTI Connext the existence of the ***DDS_Octets*** *(p. 1217) data type.*
- static **DDS_ReturnCode_t** **unregister_type** (**DDSDomainParticipant** *participant, const char *type_name="DDS::Octets")

  Allows an application to unregister the ***DDS_Octets*** *(p. 1217) data type from RTI Connext. After calling unregister_type, no further communication using this type is possible.*
- static const char * **get_type_name** ()

  Get the default name for the ***DDS_Octets*** *(p. 1217) type.*
- static void **print_data** (const **DDS_Octets** *a_data)

  <<***extension***>> *(p. 432) Print value of data type to standard out.*
- static **DDS_TypeCode** * **get_typecode** ()

  <<***extension***>> *(p. 432) Retrieves the TypeCode for the Type.*
- static **DDS_ReturnCode_t** **serialize_data_to_cdr_buffer** (char *buffer, unsigned int &length, const **DDS_Octets** *a_data)

  <<***extension***>> *(p. 432) Serializes the input sample into a CDR buffer of octets.*
- static **DDS_ReturnCode_t** **serialize_data_to_cdr_buffer_ex** (char *buffer, unsigned int &length, const **DDS_Octets** *a_data, **DDS_DataRepresentationId_t** representation)

  <<***extension***>> *(p. 432) Serializes the input sample into a buffer of octets.*
- static **DDS_ReturnCode_t** **deserialize_data_from_cdr_buffer** (**DDS_Octets** *a_data, const char *buffer, unsigned int length)

  <<***extension***>> *(p. 432) Deserializes a sample from a buffer of octets.*
- static **DDS_ReturnCode_t** **data_to_string** (**DDS_Octets** *sample, char *str, **DDS_UnsignedLong** &str_size, const **DDS_PrintFormatProperty** &property)

  <<***extension***>> *(p. 432) Get the string representation of an input sample.*

9.272.1 Detailed Description

<<***interface***>> *(p. 433)* **DDS_Octets** (p. 1217) type support.

9.272.2 Member Function Documentation

9.272.2.1 register_type()

```
static  DDS_ReturnCode_t DDSOctetsTypeSupport::register_type (
              DDSDomainParticipant * participant,
             const char * type_name = "DDS::Octets" )  [static]
```

Allows an application to communicate to RTI Connext the existence of the **DDS_Octets** (p. 1217) data type.

By default, The **DDS_Octets** (p. 1217) built-in type is automatically registered when a DomainParticipant is created using the type_name returned by **DDSOctetsTypeSupport::get_type_name** (p. 1813). Therefore, the usage of this function is optional and it is only required when the automatic built-in type registration is disabled using the participant property "dds.builtin_type.auto_register".

This method can also be used to register the same **DDSOctetsTypeSupport** (p. 1811) with a **DDSDomainParticipant** (p. 1619) using different values for the type_name.

If `register_type` is called multiple times with the same **DDSDomainParticipant** (p. 1619) and `type_name`, the second (and subsequent) registrations are ignored by the operation.

Parameters

participant	*<<**in**>> (p. 433)* the **DDSDomainParticipant** (p. 1619) to register the data type **DDS_Octets** (p. 1217) with. Cannot be NULL.
type_name	*<<**in**>> (p. 433)* the type name under with the data type **DDS_Octets** (p. 1217) is registered with the participant; this type name is used when creating a new **DDSTopic** (p. 1885). (See **DDSDomainParticipant::create_topic** (p. 1650).) The name may not be NULL or longer than 255 characters.

Exceptions

One	of the **Standard Return Codes** (p. 532), **DDS_RETCODE_PRECONDITION_NOT_MET** (p. 532) or **DDS_RETCODE_OUT_OF_RESOURCES** (p. 533).

MT Safety:

UNSAFE on the FIRST call. It is not safe for two threads to simultaneously make the first call to register a type. Subsequent calls are thread safe.

See also

DDSDomainParticipant::create_topic (p. 1650)

9.272.2.2 unregister_type()

```
static DDS_ReturnCode_t DDSOctetsTypeSupport::unregister_type (
            DDSDomainParticipant * participant,
          const char * type_name = "DDS::Octets" ) [static]
```

Allows an application to unregister the **DDS_Octets** (p. 1217) data type from RTI Connext. After calling unregister_type, no further communication using this type is possible.

Precondition

The **DDS_Octets** (p. 1217) type with `type_name` is registered with the participant and all **DDSTopic** (p. 1885) objects referencing the type have been destroyed. If the type is not registered with the participant, or if any **DDSTopic** (p. 1885) is associated with the type, the operation will fail with **DDS_RETCODE_ERROR** (p. 532).

Postcondition

All information about the type is removed from RTI Connext. No further communication using this type is possible.

Parameters

participant	<<***in***>> *(*p. *433)* the **DDSDomainParticipant** (p. 1619) to unregister the data type **DDS_Octets** (p. 1217) from. Cannot be NULL.
type_name	<<***in***>> *(*p. *433)* the type name under with the data type **DDS_Octets** (p. 1217) is registered with the participant. The name should match a name that has been previously used to register a type with the participant. Cannot be NULL.

Exceptions

One	of the **Standard Return Codes** (p. 532), **DDS_RETCODE_BAD_PARAMETER** (p. 532) or **DDS_RETCODE_ERROR** (p. 532)

MT Safety:

SAFE.

See also

DDSOctetsTypeSupport::register_type (p. 1812)

9.272.2.3 get_type_name()

```
static const char * DDSOctetsTypeSupport::get_type_name ( ) [static]
```

Get the default name for the **DDS_Octets** (p. 1217) type.

Can be used for calling **DDSOctetsTypeSupport::register_type** (p. 1812) or creating **DDSTopic** (p. 1885).

Returns

default name for the **DDS_Octets** (p. 1217) type.

See also

DDSOctetsTypeSupport::register_type (p. 1812)

DDSDomainParticipant::create_topic (p. 1650)

9.272.2.4 print_data()

```
static void DDSOctetsTypeSupport::print_data (
            const  DDS_Octets * a_data )  [static]
```

<<***extension***>> *(p. 432)* Print value of data type to standard out.

The *generated* implementation of the operation knows how to print value of a data type.

Parameters

a_data	<<***in***>> *(p. 433)* **DDS_Octets** (p. 1217) to be printed.

9.272.2.5 get_typecode()

```
static  DDS_TypeCode * DDSOctetsTypeSupport::get_typecode ( )  [static]
```

<<***extension***>> *(p. 432)* Retrieves the TypeCode for the Type.

See also

FooTypeSupport::get_typecode (p. 1991)

9.272.2.6 serialize_data_to_cdr_buffer()

```
static  DDS_ReturnCode_t DDSOctetsTypeSupport::serialize_data_to_cdr_buffer (
            char * buffer,
            unsigned int & length,
            const  DDS_Octets * a_data )  [static]
```

<<***extension***>> *(p. 432)* Serializes the input sample into a CDR buffer of octets.

See also

FooTypeSupport::serialize_data_to_cdr_buffer (p. 1989)

9.272.2.7 serialize_data_to_cdr_buffer_ex()

```
static  DDS_ReturnCode_t DDSOctetsTypeSupport::serialize_data_to_cdr_buffer_ex (
             char * buffer,
             unsigned int & length,
             const  DDS_Octets * a_data,
              DDS_DataRepresentationId_t representation )  [static]
```

<<***extension***>> *(*p. *432)* Serializes the input sample into a buffer of octets.

See also

FooTypeSupport::serialize_data_to_cdr_buffer (p. 1989)

9.272.2.8 deserialize_data_from_cdr_buffer()

```
static  DDS_ReturnCode_t DDSOctetsTypeSupport::deserialize_data_from_cdr_buffer (
              DDS_Octets * a_data,
             const char * buffer,
             unsigned int length )  [static]
```

<<***extension***>> (p. *432)* Deserializes a sample from a buffer of octets.

See also

FooTypeSupport::deserialize_data_from_cdr_buffer (p. 1990)

9.272.2.9 data_to_string()

```
static  DDS_ReturnCode_t DDSOctetsTypeSupport::data_to_string (
              DDS_Octets * sample,
             char * str,
              DDS_UnsignedLong & str_size,
             const  DDS_PrintFormatProperty & property )  [static]
```

<<***extension***>> (p. *432)* Get the string representation of an input sample.

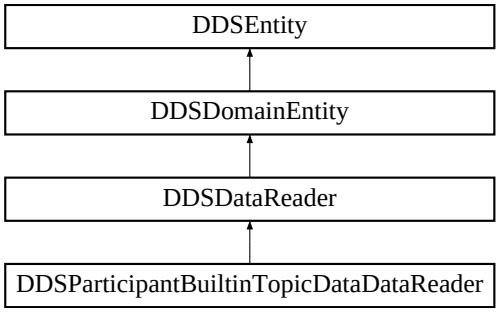
See also

FooTypeSupport::data_to_string (p. 1991)

9.273 DDSParticipantBuiltinTopicDataDataReader Class Reference

Instantiates `DataReader` < **DDS_ParticipantBuiltinTopicData** (p. 1229) > .

Inheritance diagram for DDSParticipantBuiltinTopicDataDataReader:



Additional Inherited Members

9.273.1 Detailed Description

Instantiates `DataReader` < **DDS_ParticipantBuiltinTopicData** (p. 1229) > .

DDSDataReader (p. 1556) of topic **DDS_PARTICIPANT_TOPIC_NAME** (p. 491) used for accessing **DDS_↩ ParticipantBuiltinTopicData** (p. 1229) of the remote **DDSDomainParticipant** (p. 1619).

Instantiates:

<<***generic***>> *(p. 433)* **FooDataReader** (p. 1917)

See also

DDS_ParticipantBuiltinTopicData (p. 1229)

DDS_PARTICIPANT_TOPIC_NAME (p. 491)

9.274 DDSParticipantBuiltinTopicDataTypeSupport Class Reference

Instantiates `TypeSupport` < **DDS_ParticipantBuiltinTopicData** (p. 1229) > .

9.274.1 Detailed Description

Instantiates `TypeSupport` < **DDS_ParticipantBuiltinTopicData** (p. 1229) > .

Instantiates:

<<***generic***>> *(p. 433)* **FooTypeSupport** (p. 1979)

See also

DDS_ParticipantBuiltinTopicData (p. 1229)

9.275 DDSPropertyQosPolicyHelper Class Reference

Policy helpers that facilitate management of the properties in the input policy.

Static Public Member Functions

- static **DDS_Long get_number_of_properties** (**DDS_PropertyQosPolicy** &policy)

  Gets the number of properties in the input policy.
- static **DDS_ReturnCode_t assert_property** (**DDS_PropertyQosPolicy** &policy, const char ∗name, const char ∗value, **DDS_Boolean** propagate)

  Asserts the property identified by name in the input policy.
- static **DDS_ReturnCode_t add_property** (**DDS_PropertyQosPolicy** &policy, const char ∗name, const char ∗value, **DDS_Boolean** propagate)

  Adds a new property to the input policy.
- static **DDS_ReturnCode_t assert_pointer_property** (**DDS_PropertyQosPolicy** &policy, const char ∗name, const void ∗pointer)

  Asserts the property identified by name in the input policy. Used when the property to store is a pointer.
- static **DDS_ReturnCode_t add_pointer_property** (**DDS_PropertyQosPolicy** &policy, const char ∗name, const void ∗pointer)

  Adds a new property to the input policy. Used when the property to store is a pointer.
- static struct **DDS_Property_t** ∗ **lookup_property** (**DDS_PropertyQosPolicy** &policy, const char ∗name)

  Searches for a property in the input policy given its name.
- static **DDS_ReturnCode_t remove_property** (**DDS_PropertyQosPolicy** &policy, const char ∗name)

  Removes a property from the input policy.
- static **DDS_ReturnCode_t get_properties** (**DDS_PropertyQosPolicy** &policy, struct **DDS_PropertySeq** &properties, const char ∗name_prefix)

  Retrieves a list of properties whose names match the input prefix.

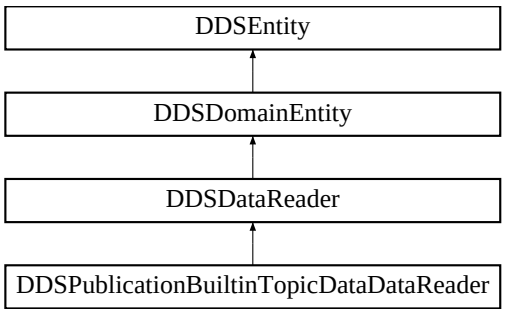
9.275.1 Detailed Description

Policy helpers that facilitate management of the properties in the input policy.

9.276 DDSPublicationBuiltinTopicDataDataReader Class Reference

Instantiates `DataReader` < **DDS_PublicationBuiltinTopicData** (p. 1264) > .

Inheritance diagram for DDSPublicationBuiltinTopicDataDataReader:



Additional Inherited Members

9.276.1 Detailed Description

Instantiates `DataReader` < **DDS_PublicationBuiltinTopicData** (p. 1264) > .

DDSDataReader (p. 1556) of topic **DDS_PUBLICATION_TOPIC_NAME** (p. 494) used for accessing **DDS_↩ PublicationBuiltinTopicData** (p. 1264) of the remote **DDSDataWriter** (p. 1589) and the associated **DDSPublisher** (p. 1818).

Instantiates:

<<***generic***>> *(p. 433)* **FooDataReader** (p. 1917)

See also

DDS_PublicationBuiltinTopicData (p. 1264)

DDS_PUBLICATION_TOPIC_NAME (p. 494)

9.277 DDSPublicationBuiltinTopicDataTypeSupport Class Reference

Instantiates `TypeSupport` < **DDS_PublicationBuiltinTopicData** (p. 1264) > .

9.277.1 Detailed Description

Instantiates `TypeSupport` < **DDS_PublicationBuiltinTopicData** (p. 1264) > .

Instantiates:

<<***generic***>> *(p. 433)* **FooTypeSupport** (p. 1979)

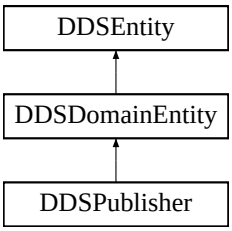
See also

DDS_PublicationBuiltinTopicData (p. 1264)

9.278 DDSPublisher Class Reference

<<***interface***>> *(p. 433)* A publisher is the object responsible for the actual dissemination of publications.

Inheritance diagram for DDSPublisher:



Public Member Functions

- virtual **DDS_ReturnCode_t get_default_datawriter_qos** (**DDS_DataWriterQos** &qos)=0

  *Copies the default **DDS_DataWriterQos** (p. 909) values into the provided **DDS_DataWriterQos** (p. 909) instance.*
- virtual **DDS_ReturnCode_t set_default_datawriter_qos** (const **DDS_DataWriterQos** &qos)=0

  *Sets the default **DDS_DataWriterQos** (p. 909) values for this publisher.*
- virtual **DDS_ReturnCode_t set_default_datawriter_qos_with_profile** (const char ∗library_name, const char ∗profile_name)=0

  *<<**extension**>> (p. 432) Set the default **DDS_DataWriterQos** (p. 909) values for this publisher based on the input XML QoS profile.*
- virtual **DDS_ReturnCode_t set_default_library** (const char ∗library_name)=0

  *<<**extension**>> (p. 432) Sets the default XML library for a **DDSPublisher** (p. 1818).*
- virtual const char ∗ **get_default_library** ()=0

  *<<**extension**>> (p. 432) Gets the default XML library associated with a **DDSPublisher** (p. 1818).*
- virtual **DDS_ReturnCode_t set_default_profile** (const char ∗library_name, const char ∗profile_name)=0

  *<<**extension**>> (p. 432) Sets the default XML profile for a **DDSPublisher** (p. 1818).*
- virtual const char ∗ **get_default_profile** ()=0

  *<<**extension**>> (p. 432) Gets the default XML profile associated with a **DDSPublisher** (p. 1818).*
- virtual const char ∗ **get_default_profile_library** ()=0

  *<<**extension**>> (p. 432) Gets the library where the default XML QoS profile is contained for a **DDSPublisher** (p. 1818).*
- virtual **DDSDataWriter** ∗ **create_datawriter** (**DDSTopic** ∗topic, const **DDS_DataWriterQos** &qos, **DDSData↩ WriterListener** ∗listener, **DDS_StatusMask** mask)=0

  *Creates a **DDSDataWriter** (p. 1589) that will be attached and belong to the **DDSPublisher** (p. 1818).*
- virtual **DDSDataWriter** ∗ **create_datawriter_with_profile** (**DDSTopic** ∗topic, const char ∗library_name, const char ∗profile_name, **DDSDataWriterListener** ∗listener, **DDS_StatusMask** mask)=0

  *<<**extension**>> (p. 432) Creates a **DDSDataWriter** (p. 1589) object using the **DDS_DataWriterQos** (p. 909) associated with the input XML QoS profile.*
- virtual **DDS_ReturnCode_t delete_datawriter** (**DDSDataWriter** ∗a_datawriter)=0

  *Deletes a **DDSDataWriter** (p. 1589) that belongs to the **DDSPublisher** (p. 1818).*
- virtual **DDSDataWriter** ∗ **lookup_datawriter** (const char ∗topic_name)=0

  *Retrieves the **DDSDataWriter** (p. 1589) for a specific **DDSTopic** (p. 1885).*
- virtual **DDS_ReturnCode_t get_all_datawriters** (DDSDataWriterSeq &writers)=0

  Retrieve all the DataWriters created from this Publisher.
- virtual **DDS_ReturnCode_t suspend_publications** ()=0

  *Indicates to RTI Connext that the application is about to make multiple modifications using **DDSDataWriter** (p. 1589) objects belonging to the **DDSPublisher** (p. 1818).*
- virtual **DDS_ReturnCode_t resume_publications** ()=0

  *Indicates to RTI Connext that the application has completed the multiple changes initiated by the previous **DDSPublisher::suspend_publications** (p. 1830).*
- virtual **DDS_ReturnCode_t begin_coherent_changes** ()=0

  *Indicates that the application will begin a coherent set of modifications using **DDSDataWriter** (p. 1589) objects attached to the **DDSPublisher** (p. 1818).*
- virtual **DDS_ReturnCode_t end_coherent_changes** ()=0

  *Terminates the coherent set initiated by the matching call to **DDSPublisher::begin_coherent_changes** (p. 1832).*
- virtual **DDSDomainParticipant** ∗ **get_participant** ()=0

  *Returns the **DDSDomainParticipant** (p. 1619) to which the **DDSPublisher** (p. 1818) belongs.*
- virtual **DDS_ReturnCode_t delete_contained_entities** ()=0

  *Deletes all the entities that were created by means of the "create" operation on the **DDSPublisher** (p. 1818).*

- virtual **DDS_ReturnCode_t copy_from_topic_qos** (**DDS_DataWriterQos** &a_datawriter_qos, const **DDS_↩ TopicQos** &a_topic_qos)=0

  *Copies the policies in the **DDS_TopicQos** (p. 1397) to the corresponding policies in the **DDS_DataWriterQos** (p. 909).*
- virtual **DDS_ReturnCode_t wait_for_acknowledgments** (const **DDS_Duration_t** &max_wait)=0

  Blocks the calling thread until all data written by the Publisher's reliable DataWriters is acknowledged, or until timeout expires.
- virtual **DDS_ReturnCode_t wait_for_asynchronous_publishing** (const **DDS_Duration_t** &max_wait)=0

  *<<**extension**>> (p. 432) Blocks the calling thread until asynchronous sending is complete.*
- virtual **DDS_ReturnCode_t set_qos** (const **DDS_PublisherQos** &qos)=0

  Sets the publisher QoS.
- virtual **DDS_ReturnCode_t set_qos_with_profile** (const char *library_name, const char *profile_name)=0

  *<<**extension**>> (p. 432) Change the QoS of this publisher using the input XML QoS profile.*
- virtual **DDS_ReturnCode_t get_qos** (**DDS_PublisherQos** &qos)=0

  Gets the publisher QoS.
- virtual **DDS_ReturnCode_t set_listener** (**DDSPublisherListener** *l, **DDS_StatusMask** mask= **DDS_↩ STATUS_MASK_ALL**)=0

  Sets the publisher listener.
- virtual **DDSPublisherListener** * **get_listener** ()=0

  Get the publisher listener.
- virtual **DDSDataWriter** * **lookup_datawriter_by_name** (const char *datawriter_name)=0

  *<<**extension**>> (p. 432) Retrieves a **DDSDataWriter** (p. 1589) contained within the **DDSPublisher** (p. 1818) the **DDSDataWriter** (p. 1589) entity name.*

9.278.1 Detailed Description

*<<**interface**>> (p. 433)* A publisher is the object responsible for the actual dissemination of publications.

QoS:

DDS_PublisherQos (p. 1276)

Listener:

DDSPublisherListener (p. 1839)

A publisher acts on the behalf of one or several **DDSDataWriter** (p. 1589) objects that belong to it. When it is informed of a change to the data associated with one of its **DDSDataWriter** (p. 1589) objects, it decides when it is appropriate to actually send the data-update message. In making this decision, it considers any extra information that goes with the data (timestamp, writer, etc.) as well as the QoS of the **DDSPublisher** (p. 1818) and the **DDSDataWriter** (p. 1589).

The following operations may be called even if the **DDSPublisher** (p. 1818) is not enabled. Other operations will fail with the value **DDS_RETCODE_NOT_ENABLED** (p. 533) if called on a disabled **DDSPublisher** (p. 1818):

- **DDSEntity::enable** (p. 1734),
- **DDSPublisher::set_qos** (p. 1836), **DDSPublisher::get_qos** (p. 1837) , **DDSPublisher::set_qos_with_profile** (p. 1837)

- **DDSPublisher::set_listener** (p. 1838), **DDSPublisher::get_listener** (p. 1838),
- **DDSEntity::get_statuscondition** (p. 1735), **DDSEntity::get_status_changes** (p. 1735)
- **DDSPublisher::create_datawriter** (p. 1826), **DDSPublisher::create_datawriter_with_profile** (p. 1827), **DDSPublisher::delete_contained_entities** (p. 1833), **DDSPublisher::delete_datawriter** (p. 1829),
- **DDSPublisher::set_default_datawriter_qos** (p. 1822), **DDSPublisher::set_default_datawriter_qos_with_profile** (p. 1823), **DDSPublisher::get_default_datawriter_qos** (p. 1821), **DDSPublisher::wait_for_acknowledgments** (p. 1835), **DDSPublisher::set_default_library** (p. 1823), **DDSPublisher::set_default_profile** (p. 1824),

See also

Operations Allowed in Listener Callbacks (p. 1796)

Examples

HelloWorld_publisher.cxx.

9.278.2 Member Function Documentation

9.278.2.1 get_default_datawriter_qos()

```
virtual DDS_ReturnCode_t DDSPublisher::get_default_datawriter_qos (
            DDS_DataWriterQos & qos ) [pure virtual]
```

Copies the default **DDS_DataWriterQos** (p. 909) values into the provided **DDS_DataWriterQos** (p. 909) instance.

The retrieved `qos` will match the set of values specified on the last successful call to **DDSPublisher::set_default_datawriter_qos** (p. 1822) or **DDSPublisher::set_default_datawriter_qos_with_profile** (p. 1823), or else, if the call was never made, the default values from its owning **DDSDomainParticipant** (p. 1619).

This method may potentially allocate memory depending on the sequences contained in some QoS policies.

MT Safety:

UNSAFE. It is not safe to retrieve the default QoS value from a **DDSPublisher** (p. 1818) while another thread may be simultaneously calling **DDSPublisher::set_default_datawriter_qos** (p. 1822).

Parameters

qos	<<***inout***>> *(p. 433)* **DDS_DataWriterQos** (p. 909) to be filled-up.

Exceptions

One	of the **Standard Return Codes** (p. 532)

See also

DDS_DATAWRITER_QOS_DEFAULT (p. 114)

DDSPublisher::create_datawriter (p. 1826)

9.278.2.2 set_default_datawriter_qos()

```
virtual DDS_ReturnCode_t DDSPublisher::set_default_datawriter_qos (
            const DDS_DataWriterQos & qos ) [pure virtual]
```

Sets the default **DDS_DataWriterQos** (p. 909) values for this publisher.

This call causes the default values inherited from the owning **DDSDomainParticipant** (p. 1619) to be overridden.

This default value will be used for newly created **DDSDataWriter** (p. 1589) if **DDS_DATAWRITER_QOS_DEFAULT** (p. 114) is specified as the `qos` parameter when **DDSPublisher::create_datawriter** (p. 1826) is called.

Precondition

The specified QoS policies must be consistent, or else the operation will have no effect and fail with **DDS_RETCODE_INCONSISTENT_POLICY** (p. 533)

MT Safety:

UNSAFE. It is not safe to set the default QoS value from a **DDSPublisher** (p. 1818) while another thread may be simultaneously calling **DDSPublisher::set_default_datawriter_qos** (p. 1822), **DDSPublisher::get_default_datawriter_qos** (p. 1821) or calling **DDSPublisher::create_datawriter** (p. 1826) with **DDS_DATAWRITER_QOS_DEFAULT** (p. 114) as the `qos` parameter.

Parameters

qos	<<***in***>> *(p. 433)* Default qos to be set. The special value **DDS_DATAREADER_QOS_DEFAULT** (p. 136) may be passed as `qos` to indicate that the default QoS should be reset back to the initial values the factory would used if **DDSPublisher::set_default_datawriter_qos** (p. 1822) had never been called.

Exceptions

One	of the **Standard Return Codes** (p. 532), or **DDS_RETCODE_INCONSISTENT_POLICY** (p. 533)

9.278.2.3 set_default_datawriter_qos_with_profile()

```
virtual  DDS_ReturnCode_t DDSPublisher::set_default_datawriter_qos_with_profile (
            const char * library_name,
            const char * profile_name )  [pure virtual]
```

<<***extension***>> *(p. 432)* Set the default **DDS_DataWriterQos** (p. 909) values for this publisher based on the input XML QoS profile.

This default value will be used for newly created **DDSDataWriter** (p. 1589) if **DDS_DATAWRITER_QOS_DEFAULT** (p. 114) is specified as the `qos` parameter when **DDSPublisher::create_datawriter** (p. 1826) is called.

Precondition

The **DDS_DataWriterQos** (p. 909) contained in the specified XML QoS profile must be consistent, or else the operation will have no effect and fail with **DDS_RETCODE_INCONSISTENT_POLICY** (p. 533)

MT Safety:

UNSAFE. It is not safe to set the default QoS value from a **DDSPublisher** (p. 1818) while another thread may be simultaneously calling **DDSPublisher::set_default_datawriter_qos** (p. 1822), **DDSPublisher::get_default_datawriter_qos** (p. 1821) or calling **DDSPublisher::create_datawriter** (p. 1826) with **DDS_DATAWRITER_QOS_DEFAULT** (p. 114) as the `qos` parameter.

Parameters

library_name	<<***in***>> *(p. 433)* Library name containing the XML QoS profile. If library_name is null RTI Connext will use the default library (see **DDSPublisher::set_default_library** (p. 1823)).
profile_name	<<***in***>> *(p. 433)* XML QoS Profile name. If profile_name is null RTI Connext will use the default profile (see **DDSPublisher::set_default_profile** (p. 1824)).

If the input profile cannot be found, the method fails with **DDS_RETCODE_ERROR** (p. 532).

Exceptions

One	of the **Standard Return Codes** (p. 532), or **DDS_RETCODE_INCONSISTENT_POLICY** (p. 533)

See also

DDS_DATAWRITER_QOS_DEFAULT (p. 114)

DDSPublisher::create_datawriter_with_profile (p. 1827)

9.278.2.4 set_default_library()

```
virtual  DDS_ReturnCode_t DDSPublisher::set_default_library (
            const char * library_name )  [pure virtual]
```

<<***extension***>> *(p. 432)* Sets the default XML library for a **DDSPublisher** (p. 1818).

This method specifies the library that will be used as the default the next time a default library is needed during a call to one of this Publisher's operations.

Any API requiring a library_name as a parameter can use null to refer to the default library.

If the default library is not set, the **DDSPublisher** (p. 1818) inherits the default from the **DDSDomainParticipant** (p. 1619) (see **DDSDomainParticipant::set_default_library** (p. 1635)).

Parameters

library_name	<<***in***>> *(p. 433)* Library name. If library_name is null any previous default is unset.

Exceptions

One	of the **Standard Return Codes** (p. 532)

See also

DDSPublisher::get_default_library (p. 1824)

9.278.2.5 get_default_library()

```
virtual const char * DDSPublisher::get_default_library ( )  [pure virtual]
```

<<***extension***>> *(p. 432)* Gets the default XML library associated with a **DDSPublisher** (p. 1818).

Returns

The default library or null if the default library was not set.

See also

DDSPublisher::set_default_library (p. 1823)

9.278.2.6 set_default_profile()

```
virtual  DDS_ReturnCode_t DDSPublisher::set_default_profile (
            const char * library_name,
            const char * profile_name )  [pure virtual]
```

<<***extension***>> *(p. 432)* Sets the default XML profile for a **DDSPublisher** (p. 1818).

This method specifies the profile that will be used as the default the next time a default Publisher profile is needed during a call to one of this Publisher's operations. When calling a **DDSPublisher** (p. 1818) method that requires a `profile_name` parameter, you can use NULL to refer to the default profile. (This same information applies to setting a default library.)

If the default profile is not set, the **DDSPublisher** (p. 1818) inherits the default from the **DDSDomainParticipant** (p. 1619) (see **DDSDomainParticipant::set_default_profile** (p. 1636)).

This method does not set the default QoS for **DDSDataWriter** (p. 1589) objects created by the **DDSPublisher** (p. 1818); for this functionality, use **DDSPublisher::set_default_datawriter_qos_with_profile** (p. 1823) (you may pass in NULL after having called **set_default_profile()** (p. 1824)).

This method does not set the default QoS for newly created Publishers; for this functionality, use **DDSDomainParticipant::set_default_publisher_qos_with_profile** (p. 1640).

Parameters

library_name	<<***in***>> *(p. 433)* The library name containing the profile.
profile_name	<<***in***>> *(p. 433)* The profile name. If profile_name is null any previous default is unset.

Exceptions

One	of the **Standard Return Codes** (p. 532)

See also

DDSPublisher::get_default_profile (p. 1825)

DDSPublisher::get_default_profile_library (p. 1826)

9.278.2.7 get_default_profile()

```
virtual const char * DDSPublisher::get_default_profile ( )  [pure virtual]
```

<<***extension***>> *(p. 432)* Gets the default XML profile associated with a **DDSPublisher** (p. 1818).

Returns

The default profile or null if the default profile was not set.

See also

DDSPublisher::set_default_profile (p. 1824)

9.278.2.8 get_default_profile_library()

```
virtual const char * DDSPublisher::get_default_profile_library ( )  [pure virtual]
```

<<***extension***>> *(p. 432)* Gets the library where the default XML QoS profile is contained for a **DDSPublisher** (p. 1818).

The default profile library is automatically set when **DDSPublisher::set_default_profile** (p. 1824) is called.

This library can be different than the **DDSPublisher** (p. 1818) default library (see **DDSPublisher::get_default_library** (p. 1824)).

Returns

The default profile library or null if the default profile was not set.

See also

DDSPublisher::set_default_profile (p. 1824)

9.278.2.9 create_datawriter()

```
virtual  DDSDataWriter * DDSPublisher::create_datawriter (
             DDSTopic * topic,
            const  DDS_DataWriterQos & qos,
             DDSDataWriterListener * listener,
             DDS_StatusMask mask )  [pure virtual]
```

Creates a **DDSDataWriter** (p. 1589) that will be attached and belong to the **DDSPublisher** (p. 1818).

For each application-defined type, **Foo** (p. 1916), there is an implied, auto-generated class **FooDataWriter** (p. 1944) that extends **DDSDataWriter** (p. 1589) and contains the operations to write data of type **Foo** (p. 1916).

Note that a common application pattern to construct the QoS for the **DDSDataWriter** (p. 1589) is to:

- Retrieve the QoS policies on the associated **DDSTopic** (p. 1885) by means of the **DDSTopic::get_qos** (p. 1889) operation.
- Retrieve the default **DDSDataWriter** (p. 1589) qos by means of the **DDSPublisher::get_default_datawriter_qos** (p. 1821) operation.
- Combine those two QoS policies (for example, using **DDSPublisher::copy_from_topic_qos** (p. 1834)) and selectively modify policies as desired.

When a **DDSDataWriter** (p. 1589) is created, only those transports already registered are available to the **DDSData↩ Writer** (p. 1589). See **Built-in Transport Plugins** (p. 182) for details on when a builtin transport is registered.

Precondition

If publisher is enabled, topic must have been enabled. Otherwise, this operation will fail and no **DDSDataWriter** (p. 1589) will be created.

The given **DDSTopic** (p. 1885) must have been created from the same participant as this publisher. If it was created from a different participant, this method will fail.

MT Safety:

UNSAFE. If **DDS_DATAWRITER_QOS_DEFAULT** (p. 114) is used for the `qos` parameter, it is not safe to create the datawriter while another thread may be simultaneously calling **DDSPublisher::set_default_datawriter_qos** (p. 1822).

Parameters

topic	*<<**in**>> (p. 433)* The **DDSTopic** (p. 1885) that the **DDSDataWriter** (p. 1589) will be associated with. Cannot be NULL.
qos	*<<**in**>> (p. 433)* QoS to be used for creating the new **DDSDataWriter** (p. 1589). The special value **DDS_DATAWRITER_QOS_DEFAULT** (p. 114) can be used to indicate that the **DDSDataWriter** (p. 1589) should be created with the default **DDS_DataWriterQos** (p. 909) set in the **DDSPublisher** (p. 1818). The special value **DDS_DATAWRITER_QOS_USE_TOPIC_QOS** (p. 114) can be used to indicate that the **DDSDataWriter** (p. 1589) should be created with the combination of the default **DDS_DataWriterQos** (p. 909) set on the **DDSPublisher** (p. 1818) and the **DDS_TopicQos** (p. 1397) of the **DDSTopic** (p. 1885).
listener	*<<**in**>> (p. 433)* The listener of the **DDSDataWriter** (p. 1589).
mask	*<<**in**>> (p. 433).* Changes of communication status to be invoked on the listener. See **DDS_StatusMask** (p. 537).

Returns

A **DDSDataWriter** (p. 1589) of a derived class specific to the data type associated with the **DDSTopic** (p. 1885) or NULL if an error occurred.

See also

FooDataWriter (p. 1944)

Specifying QoS on entities (p. 555) for information on setting QoS before entity creation

DDS_DataWriterQos (p. 909) for rules on consistency among QoS

DDS_DATAWRITER_QOS_DEFAULT (p. 114)

DDS_DATAWRITER_QOS_USE_TOPIC_QOS (p. 114)

DDSPublisher::create_datawriter_with_profile (p. 1827)

DDSPublisher::get_default_datawriter_qos (p. 1821)

DDSTopic::set_qos (p. 1887)

DDSPublisher::copy_from_topic_qos (p. 1834)

DDSDataWriter::set_listener (p. 1607)

9.278.2.10 create_datawriter_with_profile()

```
virtual DDSDataWriter * DDSPublisher::create_datawriter_with_profile (
            DDSTopic * topic,
          const char * library_name,
          const char * profile_name,
            DDSDataWriterListener * listener,
            DDS_StatusMask mask ) [pure virtual]
```

<<***extension***>> *(p. 432)* Creates a **DDSDataWriter** (p. 1589) object using the **DDS_DataWriterQos** (p. 909) associated with the input XML QoS profile.

The **DDSDataWriter** (p. 1589) will be attached and belong to the **DDSPublisher** (p. 1818).

For each application-defined type, **Foo** (p. 1916), there is an implied, auto-generated class **FooDataWriter** (p. 1944) that extends **DDSDataWriter** (p. 1589) and contains the operations to write data of type **Foo** (p. 1916).

When a **DDSDataWriter** (p. 1589) is created, only those transports already registered are available to the **DDSDataWriter** (p. 1589). See **Built-in Transport Plugins** (p. 182) for details on when a builtin transport is registered.

Precondition

If publisher is enabled, topic must have been enabled. Otherwise, this operation will fail and no **DDSDataWriter** (p. 1589) will be created.

The given **DDSTopic** (p. 1885) must have been created from the same participant as this publisher. If it was created from a different participant, this method will return NULL.

Parameters

topic	<<***in***>> *(p. 433)* The **DDSTopic** (p. 1885) that the **DDSDataWriter** (p. 1589) will be associated with. Cannot be NULL.
library_name	<<***in***>> *(p. 433)* Library name containing the XML QoS profile. If library_name is null RTI Connext will use the default library (see **DDSPublisher::set_default_library** (p. 1823)).
profile_name	<<***in***>> *(p. 433)* XML QoS Profile name. If profile_name is null RTI Connext will use the default profile (see **DDSPublisher::set_default_profile** (p. 1824)).
listener	<<***in***>> *(p. 433)* The listener of the **DDSDataWriter** (p. 1589).
mask	<<***in***>> *(p. 433).* Changes of communication status to be invoked on the listener. See **DDS_StatusMask** (p. 537).

Returns

A **DDSDataWriter** (p. 1589) of a derived class specific to the data type associated with the **DDSTopic** (p. 1885) or NULL if an error occurred.

See also

FooDataWriter (p. 1944)

Specifying QoS on entities (p. 555) for information on setting QoS before entity creation

DDS_DataWriterQos (p. 909) for rules on consistency among QoS

DDSPublisher::create_datawriter (p. 1826)

DDSPublisher::get_default_datawriter_qos (p. 1821)

DDSTopic::set_qos (p. 1887)

DDSPublisher::copy_from_topic_qos (p. 1834)

DDSDataWriter::set_listener (p. 1607)

Examples

HelloWorld_publisher.cxx.

9.278.2.11 delete_datawriter()

```
virtual DDS_ReturnCode_t DDSPublisher::delete_datawriter (
            DDSDataWriter * a_datawriter ) [pure virtual]
```

Deletes a **DDSDataWriter** (p. 1589) that belongs to the **DDSPublisher** (p. 1818).

The deletion of the **DDSDataWriter** (p. 1589) will automatically unregister all instances.

Precondition

If the **DDSDataWriter** (p. 1589) does not belong to the **DDSPublisher** (p. 1818), the operation will fail with **DDS_RETCODE_PRECONDITION_NOT_MET** (p. 532).

Postcondition

Listener installed on the **DDSDataWriter** (p. 1589) will not be called after this method completes successfully.

MT Safety:

UNSAFE. It is not safe to delete an entity while another thread may be simultaneously calling an API that uses the entity.

Parameters

a_datawriter	<<***in***>> *(p. 433)* The **DDSDataWriter** (p. 1589) to be deleted.

Exceptions

One	of the **Standard Return Codes** (p. 532) or **DDS_RETCODE_PRECONDITION_NOT_MET** (p. 532).

9.278.2.12 lookup_datawriter()

```
virtual DDSDataWriter * DDSPublisher::lookup_datawriter (
            const char * topic_name ) [pure virtual]
```

Retrieves the **DDSDataWriter** (p. 1589) for a specific **DDSTopic** (p. 1885).

This returned **DDSDataWriter** (p. 1589) is either enabled or disabled.

If more than one **DDSDataWriter** (p. 1589) is attached to the **DDSPublisher** (p. 1818) with the same `topic_name`, then this operation may return any one of them.

MT Safety:

UNSAFE. It is not safe to lookup a **DDSDataWriter** (p. 1589) in one thread while another thread is simultaneously creating or destroying that **DDSDataWriter** (p. 1589).

Parameters

topic_name	<<***in***>> *(p. 433)* Name of the **DDSTopic** (p. 1885) associated with the **DDSDataWriter** (p. 1589) that is to be looked up. Cannot be NULL.

Returns

A **DDSDataWriter** (p. 1589) that belongs to the **DDSPublisher** (p. 1818) attached to the **DDSTopic** (p. 1885) with `topic_name`. If no such **DDSDataWriter** (p. 1589) exists, this operation returns NULL.

9.278.2.13 get_all_datawriters()

```
virtual DDS_ReturnCode_t DDSPublisher::get_all_datawriters (
            DDSDataWriterSeq & writers ) [pure virtual]
```

Retrieve all the DataWriters created from this Publisher.

Parameters

writers	<<***inout***>> *(p. 433)* Sequence where the DataWriters will be added

Exceptions

One	of the **Standard Return Codes** (p. 532)

9.278.2.14 suspend_publications()

```
virtual DDS_ReturnCode_t DDSPublisher::suspend_publications ( ) [pure virtual]
```

Indicates to RTI Connext that the application is about to make multiple modifications using **DDSDataWriter** (p. 1589) objects belonging to the **DDSPublisher** (p. 1818).

It is a **hint** to RTI Connext so it can optimize its performance by e.g., holding the dissemination of the modifications and then batching them.

The use of this operation must be matched by a corresponding call to **DDSPublisher::resume_publications** (p. 1831) indicating that the set of modifications has completed.

If the **DDSPublisher** (p. 1818) is deleted before **DDSPublisher::resume_publications** (p. 1831) is called, any suspended updates yet to be published will be discarded.

RTI Connext is not required and does not currently make use of this hint in any way. However, similar results can be achieved by using *asynchronous publishing*. Combined with **DDSFlowController** (p. 1736), **DDS_ASYNCHRONOUS↩ _PUBLISH_MODE_QOS** (p. 629) **DDSDataWriter** (p. 1589) instances allow the user even finer control of traffic shaping and sample coalescing.

Exceptions

One	of the **Standard Return Codes** (p. 532) or **DDS_RETCODE_NOT_ENABLED** (p. 533).

See also

DDSFlowController (p. 1736)
DDSFlowController::trigger_flow (p. 1738)
DDS_ON_DEMAND_FLOW_CONTROLLER_NAME (p. 127)
DDS_PublishModeQosPolicy (p. 1279)

9.278.2.15 resume_publications()

```
virtual DDS_ReturnCode_t DDSPublisher::resume_publications ( ) [pure virtual]
```

Indicates to RTI Connext that the application has completed the multiple changes initiated by the previous **DDSPublisher::suspend_publications** (p. 1830).

This is a **hint** to RTI Connext that can be used for example, to batch all the modifications made since the **DDSPublisher::suspend_publications** (p. 1830).

RTI Connext is not required and does not currently make use of this hint in any way. However, similar results can be achieved by using *asynchronous publishing*. Combined with **DDSFlowController** (p. 1736), **DDS_ASYNCHRONOUS↩ _PUBLISH_MODE_QOS** (p. 629) **DDSDataWriter** (p. 1589) instances allow the user even finer control of traffic shaping and sample coalescing.

Precondition

A call to **DDSPublisher::resume_publications** (p. 1831) must match a previous call to **DDSPublisher↩ ::suspend_publications** (p. 1830). Otherwise the operation will fail with **DDS_RETCODE_PRECONDITION↩ _NOT_MET** (p. 532).

Exceptions

One	of the **Standard Return Codes** (p. 532) or **DDS_RETCODE_PRECONDITION_NOT_MET** (p. 532) or **DDS_RETCODE_NOT_ENABLED** (p. 533).

See also

DDSFlowController (p. 1736)

DDSFlowController::trigger_flow (p. 1738)

DDS_ON_DEMAND_FLOW_CONTROLLER_NAME (p. 127)

DDS_PublishModeQosPolicy (p. 1279)

9.278.2.16 begin_coherent_changes()

```
virtual DDS_ReturnCode_t DDSPublisher::begin_coherent_changes ( ) [pure virtual]
```

Indicates that the application will begin a coherent set of modifications using **DDSDataWriter** (p. 1589) objects attached to the **DDSPublisher** (p. 1818).

A 'coherent set' is a set of modifications that must be propagated in such a way that they are interpreted at the receiver's side as a consistent set of modifications; that is, the receiver will only be able to access the data after all the modifications in the set are available at the receiver end.

A connectivity change may occur in the middle of a set of coherent changes; for example, the set of partitions used by the **DDSPublisher** (p. 1818) or one of its subscribers (**DDSSubscriber** (p. 1860)) may change, a late-joining **DDSDataReader** (p. 1556) may appear on the network, or a communication failure may occur. In the event that such a change prevents an entity from receiving the entire set of coherent changes, that entity must behave as if it had received none of the set.

These calls can be nested. In that case, the coherent set terminates only with the last call to **DDSPublisher::end_coherent_changes** (p. 1833). Publisher's samples (samples published by any of the DataWriters within the Publisher) that are not published within a begin_coherent_changes/end_coherent_changes block will not be provided to the DataReaders as a set.

The support for coherent changes enables a publishing application to change the value of several data-instances that could belong to the same or different topics and have those changes be seen *atomically* by the readers. This is useful in cases where the values are inter-related (for example, if there are two data-instances representing the altitude and velocity vector of the same aircraft and both are changed, it may be useful to communicate those values in a way the reader can see both together; otherwise, it may, e.g., erroneously interpret that the aircraft is on a collision course).

Note

Coherent sets don't apply to Topic Queries. If a **DDSTopicQuery** (p. 1895) selects only a subset of samples that was published as a coherent set, the subscribing application will receive them regardless of their membership to the coherent set.

Exceptions

One	of the **Standard Return Codes** (p. 532) or **DDS_RETCODE_NOT_ENABLED** (p. 533).

See also

DDS_PresentationQosPolicy (p. 1249)

9.278.2.17 end_coherent_changes()

```
virtual DDS_ReturnCode_t DDSPublisher::end_coherent_changes ( ) [pure virtual]
```

Terminates the coherent set initiated by the matching call to **DDSPublisher::begin_coherent_changes** (p. 1832).

Precondition

If there is no matching call to **DDSPublisher::begin_coherent_changes** (p. 1832) the operation will fail with **DDS_RETCODE_PRECONDITION_NOT_MET** (p. 532).

Exceptions

One	of the **Standard Return Codes** (p. 532), **DDS_RETCODE_PRECONDITION_NOT_MET** (p. 532) or **DDS_RETCODE_NOT_ENABLED** (p. 533).

9.278.2.18 get_participant()

```
virtual DDSDomainParticipant * DDSPublisher::get_participant ( ) [pure virtual]
```

Returns the **DDSDomainParticipant** (p. 1619) to which the **DDSPublisher** (p. 1818) belongs.

Returns

the **DDSDomainParticipant** (p. 1619) to which the **DDSPublisher** (p. 1818) belongs.

9.278.2.19 delete_contained_entities()

```
virtual DDS_ReturnCode_t DDSPublisher::delete_contained_entities ( ) [pure virtual]
```

Deletes all the entities that were created by means of the "create" operation on the **DDSPublisher** (p. 1818).

Deletes all contained **DDSDataWriter** (p. 1589) objects. Once **DDSPublisher::delete_contained_entities** (p. 1833) completes successfully, the application may delete the **DDSPublisher** (p. 1818), knowing that it has no contained **DDSDataWriter** (p. 1589) objects.

The operation will fail with **DDS_RETCODE_PRECONDITION_NOT_MET** (p. 532) if any of the contained entities is in a state where it cannot be deleted.

MT Safety:

UNSAFE. It is not safe to delete an entity while another thread may be simultaneously calling an API that uses the entity.

Exceptions

One	of the **Standard Return Codes** (p. 532) or **DDS_RETCODE_PRECONDITION_NOT_MET** (p. 532).

9.278.2.20 copy_from_topic_qos()

```
virtual DDS_ReturnCode_t DDSPublisher::copy_from_topic_qos (
            DDS_DataWriterQos & a_datawriter_qos,
          const DDS_TopicQos & a_topic_qos ) [pure virtual]
```

Copies the policies in the **DDS_TopicQos** (p. 1397) to the corresponding policies in the **DDS_DataWriterQos** (p. 909).

Copies the policies in the **DDS_TopicQos** (p. 1397) to the corresponding policies in the **DDS_DataWriterQos** (p. 909) (replacing values in the **DDS_DataWriterQos** (p. 909), if present).

This is a "convenience" operation most useful in combination with the operations **DDSPublisher::get_default_↩ datawriter_qos** (p. 1821) and **DDSTopic::get_qos** (p. 1889). The operation **DDSPublisher::copy_from_topic_qos** (p. 1834) can be used to merge the **DDSDataWriter** (p. 1589) default QoS policies with the corresponding ones on the **DDSTopic** (p. 1885). The resulting QoS can then be used to create a new **DDSDataWriter** (p. 1589), or set its QoS.

This operation does not check the resulting **DDS_DataWriterQos** (p. 909) for consistency. This is because the 'merged' **DDS_DataWriterQos** (p. 909) may not be the final one, as the application can still modify some policies prior to applying the policies to the **DDSDataWriter** (p. 1589).

Parameters

a_datawriter_qos	*<<**inout**>> (p. 433)* **DDS_DataWriterQos** (p. 909) to be filled-up.
a_topic_qos	*<<**in**>> (p. 433)* **DDS_TopicQos** (p. 1397) to be merged with **DDS_DataWriterQos** (p. 909).

Exceptions

One	of the **Standard Return Codes** (p. 532)

9.278.2.21 wait_for_acknowledgments()

```
virtual DDS_ReturnCode_t DDSPublisher::wait_for_acknowledgments (
            const DDS_Duration_t & max_wait ) [pure virtual]
```

Blocks the calling thread until all data written by the Publisher's reliable DataWriters is acknowledged, or until timeout expires.

This operation blocks the calling thread until either all data written by the reliable DataWriters entities is acknowledged by (a) all reliable **DDSDataReader** (p. 1556) entities that are matched and alive and (b) by all required subscriptions, or until the duration specified by the `max_wait` parameter elapses, whichever happens first. A successful completion indicates that all the samples written have been acknowledged; a timeout indicates that max_wait elapsed before all the data was acknowledged.

Note that if a thread is blocked in the call to this operation on a **DDSPublisher** (p. 1818) and a different thread writes new samples on any of the reliable DataWriters that belong to this Publisher, the new samples must be acknowledged before unblocking the thread that is waiting on this operation.

If none of the **DDSDataWriter** (p. 1589) instances have **DDS_ReliabilityQosPolicy** (p. 1295) kind set to RELIABLE, the operation will complete successfully.

Parameters

max_wait	<<***in***>> *(p. 433)* Specifies maximum time to wait for acknowledgements **DDS_Duration_t** (p. 1012) .

Exceptions

One	of the **Standard Return Codes** (p. 532), **DDS_RETCODE_NOT_ENABLED** (p. 533), **DDS_RETCODE_TIMEOUT** (p. 533)

9.278.2.22 wait_for_asynchronous_publishing()

```
virtual DDS_ReturnCode_t DDSPublisher::wait_for_asynchronous_publishing (
            const DDS_Duration_t & max_wait ) [pure virtual]
```

<<***extension***>> *(p. 432)* Blocks the calling thread until asynchronous sending is complete.

This operation blocks the calling thread (up to `max_wait`) until all data written by the asynchronous **DDSDataWriter** (p. 1589) entities is sent and acknowledged (if reliable) by all matched **DDSDataReader** (p. 1556) entities. A successful

completion indicates that all the samples written have been sent and acknowledged where applicable; if it times out, this indicates that `max_wait` elapsed before all the data was sent and/or acknowledged.

In other words, this guarantees that sending to best effort **DDSDataReader** (p. 1556) is complete in addition to what **DDSPublisher::wait_for_acknowledgments** (p. 1835) provides.

If none of the **DDSDataWriter** (p. 1589) instances have **DDS_PublishModeQosPolicy::kind** (p. 1281) set to **DDS_↩ ASYNCHRONOUS_PUBLISH_MODE_QOS** (p. 629), the operation will complete immediately , with **DDS_RETCODE↩ _OK** (p. 532).

Parameters

max_wait	*<<**in**>> (p. 433)* Specifies maximum time to wait for acknowledgements **DDS_Duration_t** (p. 1012).

Exceptions

One	of the **Standard Return Codes** (p. 532), **DDS_RETCODE_NOT_ENABLED** (p. 533), **DDS_RETCODE_TIMEOUT** (p. 533)

9.278.2.23 set_qos()

```
virtual DDS_ReturnCode_t DDSPublisher::set_qos (
            const DDS_PublisherQos & qos ) [pure virtual]
```

Sets the publisher QoS.

This operation modifies the QoS of the **DDSPublisher** (p. 1818).

The **DDS_PublisherQos::group_data** (p. 1278), **DDS_PublisherQos::partition** (p. 1278) and **DDS_PublisherQos↩ ::entity_factory** (p. 1278) can be changed. The other policies are immutable.

Parameters

qos	*<<**in**>> (p. 433)* **DDS_PublisherQos** (p. 1276) to be set to. Policies must be consistent. Immutable policies cannot be changed after **DDSPublisher** (p. 1818) is enabled. The special value **DDS_PUBLISHER_QOS_DEFAULT** (p. 59) can be used to indicate that the QoS of the **DDSPublisher** (p. 1818) should be changed to match the current default **DDS_PublisherQos** (p. 1276) set in the **DDSDomainParticipant** (p. 1619).

Exceptions

One	of the **Standard Return Codes** (p. 532), **DDS_RETCODE_IMMUTABLE_POLICY** (p. 533), or **DDS_RETCODE_INCONSISTENT_POLICY** (p. 533).

See also

DDS_PublisherQos (p. 1276) for rules on consistency among QoS

set_qos (abstract) (p. 1732)

Operations Allowed in Listener Callbacks (p. 1796)

9.278.2.24 set_qos_with_profile()

```
virtual  DDS_ReturnCode_t DDSPublisher::set_qos_with_profile (
             const char * library_name,
             const char * profile_name )  [pure virtual]
```

<<***extension***>> *(p. 432)* Change the QoS of this publisher using the input XML QoS profile.

This operation modifies the QoS of the **DDSPublisher** (p. 1818).

The **DDS_PublisherQos::group_data** (p. 1278), **DDS_PublisherQos::partition** (p. 1278) and **DDS_PublisherQos::entity_factory** (p. 1278) can be changed. The other policies are immutable.

Parameters

library_name	<<***in***>> *(p. 433)* Library name containing the XML QoS profile. If library_name is null RTI Connext will use the default library (see **DDSPublisher::set_default_library** (p. 1823)).
profile_name	<<***in***>> *(p. 433)* XML QoS Profile name. If profile_name is null RTI Connext will use the default profile (see **DDSPublisher::set_default_profile** (p. 1824)).

Exceptions

One	of the **Standard Return Codes** (p. 532), **DDS_RETCODE_IMMUTABLE_POLICY** (p. 533), or **DDS_RETCODE_INCONSISTENT_POLICY** (p. 533).

See also

DDS_PublisherQos (p. 1276) for rules on consistency among QoS

Operations Allowed in Listener Callbacks (p. 1796)

9.278.2.25 get_qos()

```
virtual  DDS_ReturnCode_t DDSPublisher::get_qos (
             DDS_PublisherQos & qos )  [pure virtual]
```

Gets the publisher QoS.

This method may potentially allocate memory depending on the sequences contained in some QoS policies.

Parameters

qos	<<***in***>> *(*p. *433)* **DDS_PublisherQos** (p. 1276) to be filled in.

Exceptions

One	of the **Standard Return Codes** (p. 532)

See also

get_qos (abstract) (p. 1733)

9.278.2.26 set_listener()

```
virtual DDS_ReturnCode_t DDSPublisher::set_listener (
              DDSPublisherListener * l,
              DDS_StatusMask mask = DDS_STATUS_MASK_ALL ) [pure virtual]
```

Sets the publisher listener.

Parameters

l	<<***in***>> *(*p. *433)* **DDSPublisherListener** (p. 1839) to set to.
mask	<<***in***>> *(*p. *433)* **DDS_StatusMask** (p. 537) associated with the **DDSPublisherListener** (p. 1839).

Exceptions

One	of the **Standard Return Codes** (p. 532)

See also

set_listener (abstract) (p. 1733)

9.278.2.27 get_listener()

```
virtual DDSPublisherListener * DDSPublisher::get_listener ( ) [pure virtual]
```

Get the publisher listener.

Returns

DDSPublisherListener (p. 1839) of the **DDSPublisher** (p. 1818).

See also

get_listener (abstract) (p. 1734)

9.278.2.28 lookup_datawriter_by_name()

```
virtual DDSDataWriter * DDSPublisher::lookup_datawriter_by_name (
            const char * datawriter_name ) [pure virtual]
```

<<***extension***>> *(p. 432)* Retrieves a **DDSDataWriter** (p. 1589) contained within the **DDSPublisher** (p. 1818) the **DDSDataWriter** (p. 1589) entity name.

Every **DDSDataWriter** (p. 1589) in the system has an entity name which is configured and stored in the <<***extension***>> *(p. 432)* EntityName policy, **ENTITY_NAME** (p. 601).

This operation retrieves the **DDSDataWriter** (p. 1589) within the **DDSPublisher** (p. 1818) whose name matches the one specified. If there are several **DDSDataWriter** (p. 1589) with the same name within the **DDSPublisher** (p. 1818), the operation returns the first matching occurrence.

Parameters

datawriter_name	<<***in***>> *(p. 433)* Entity name of the **DDSDataWriter** (p. 1589).

Returns

The first **DDSDataWriter** (p. 1589) found with the specified name or NULL if it is not found.

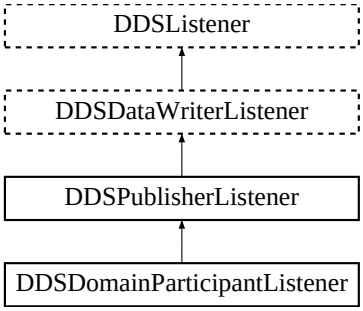
See also

DDSDomainParticipant::lookup_datawriter_by_name (p. 1691)

9.279 DDSPublisherListener Class Reference

<<***interface***>> *(p. 433)* **DDSListener** (p. 1794) for **DDSPublisher** (p. 1818) status.

Inheritance diagram for DDSPublisherListener:



Additional Inherited Members

9.279.1 Detailed Description

<<***interface***>> *(p. 433)* **DDSListener** (p. 1794) for **DDSPublisher** (p. 1818) status.

Entity:

DDSPublisher (p. 1818)

Status:

DDS_LIVELINESS_LOST_STATUS (p. 542), **DDS_LivelinessLostStatus** (p. 1179);
DDS_OFFERED_DEADLINE_MISSED_STATUS (p. 539), **DDS_OfferedDeadlineMissedStatus** (p. 1219);
DDS_OFFERED_INCOMPATIBLE_QOS_STATUS (p. 540), **DDS_OfferedIncompatibleQosStatus** (p. 1221);
DDS_PUBLICATION_MATCHED_STATUS (p. 543), **DDS_PublicationMatchedStatus** (p. 1273);
DDS_RELIABLE_READER_ACTIVITY_CHANGED_STATUS (p. 545), **DDS_ReliableReaderActivityChangedStatus** (p. 1300);
DDS_RELIABLE_WRITER_CACHE_CHANGED_STATUS (p. 545), **DDS_ReliableWriterCacheChangedStatus** (p. 1302)

See also

DDSListener (p. 1794)

Status Kinds (p. 533)

Operations Allowed in Listener Callbacks (p. 1796)

9.280 DDSPublisherSeq Class Reference

Declares IDL `sequence` < **DDSPublisher** (p. 1818) > .

9.280.1 Detailed Description

Declares IDL `sequence` < **DDSPublisher** (p. 1818) > .

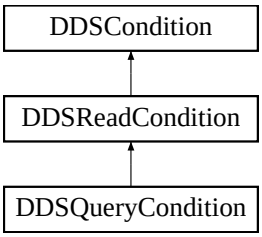
See also

FooSeq (p. 1965)

9.281 DDSQueryCondition Class Reference

<<***interface***>> *(*p. *433)* These are specialised **DDSReadCondition** (p. 1842) objects that allow the application to also specify a filter on the locally available data.

Inheritance diagram for DDSQueryCondition:



Public Member Functions

- virtual const char * **get_query_expression** ()=0

  Retrieves the query expression.
- virtual **DDS_ReturnCode_t** **get_query_parameters** (**DDS_StringSeq** &query_parameters)=0

  Retrieves the query parameters.
- virtual **DDS_ReturnCode_t** **set_query_parameters** (const **DDS_StringSeq** &query_parameters)=0

  Sets the query parameters.

9.281.1 Detailed Description

<<***interface***>> *(*p. *433)* These are specialised **DDSReadCondition** (p. 1842) objects that allow the application to also specify a filter on the locally available data.

Each query condition filter is composed of a **DDSReadCondition** (p. 1842) state filter and a content filter expressed as a `query_expression` and `query_parameters`.

The query (`query_expression`) is similar to an SQL WHERE clause and can be parameterised by arguments that are dynamically changeable by the **set_query_parameters()** (p. 1842) operation.

Two query conditions that have the same `query_expression` will require unique query condition content filters if their `query_paramters` differ. Query conditions that differ only in their state masks will share the same query condition content filter.

Queries and Filters Syntax (p. 184) describes the syntax of `query_expression` and `query_parameters`.

9.281.2 Member Function Documentation

9.281.2.1 get_query_expression()

```
virtual const char * DDSQueryCondition::get_query_expression ( )  [pure virtual]
```

Retrieves the query expression.

9.281.2.2 get_query_parameters()

```
virtual  DDS_ReturnCode_t DDSQueryCondition::get_query_parameters (
             DDS_StringSeq & query_parameters )  [pure virtual]
```

Retrieves the query parameters.

Parameters

query_parameters	<<***inout***>> *(p. 433)* the query parameters are returned here. The memory for the strings in this sequence is managed according to the conventions described in **String Conventions** (p. 758). In particular, be careful to avoid a situation in which RTI Connext allocates a string on your behalf and you then reuse that string in such a way that RTI Connext believes it to have more memory allocated to it than it actually does.

9.281.2.3 set_query_parameters()

```
virtual  DDS_ReturnCode_t DDSQueryCondition::set_query_parameters (
            const  DDS_StringSeq & query_parameters )  [pure virtual]
```

Sets the query parameters.

Parameters

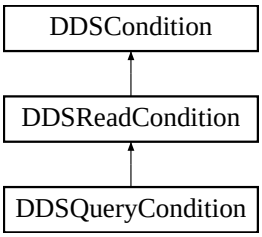
query_parameters	<<***in***>> *(p. 433)* the new query parameters

9.282 DDSReadCondition Class Reference

<<***interface***>> *(p. 433)* Conditions specifically dedicated to read operations and attached to one **DDSDataReader**

(p. 1556).

Inheritance diagram for DDSReadCondition:



Public Member Functions

- virtual **DDS_SampleStateMask get_sample_state_mask** ()=0

  Retrieves the set of `sample_states` *for the condition.*
- virtual **DDS_ViewStateMask get_view_state_mask** ()=0

  Retrieves the set of `view_states` *for the condition.*
- virtual **DDS_InstanceStateMask get_instance_state_mask** ()=0

  Retrieves the set of `instance_states` *for the condition.*
- virtual **DDS_StreamKindMask get_stream_kind_mask** ()=0

  Retrieves the set of `instance_states` *for the condition.*
- virtual **DDSDataReader** * **get_datareader** ()=0

  *Returns the **DDSDataReader** (p. 1556) associated with the **DDSReadCondition** (p. 1842).*

9.282.1 Detailed Description

<<***interface***>> *(p. 433)* Conditions specifically dedicated to read operations and attached to one **DDSDataReader** (p. 1556).

DDSReadCondition (p. 1842) objects allow an application to specify the data samples it is interested in (by specifying the desired `sample_states`, `view_states` as well as `instance_states` in **FooDataReader::read** (p. 1919) and **FooDataReader::take** (p. 1920) variants.

This allows RTI Connext to enable the condition only when suitable information is available. They are to be used in conjunction with a WaitSet as normal conditions.

More than one **DDSReadCondition** (p. 1842) may be attached to the same **DDSDataReader** (p. 1556).

Examples

HelloWorld_subscriber.cxx.

9.282.2 Member Function Documentation

9.282.2.1 get_sample_state_mask()

```
virtual DDS_SampleStateMask DDSReadCondition::get_sample_state_mask ( ) [pure virtual]
```

Retrieves the set of `sample_states` for the condition.

9.282.2.2 get_view_state_mask()

```
virtual DDS_ViewStateMask DDSReadCondition::get_view_state_mask ( ) [pure virtual]
```

Retrieves the set of `view_states` for the condition.

9.282.2.3 get_instance_state_mask()

```
virtual DDS_InstanceStateMask DDSReadCondition::get_instance_state_mask ( ) [pure virtual]
```

Retrieves the set of `instance_states` for the condition.

9.282.2.4 get_stream_kind_mask()

```
virtual DDS_StreamKindMask DDSReadCondition::get_stream_kind_mask ( ) [pure virtual]
```

Retrieves the set of `instance_states` for the condition.

9.282.2.5 get_datareader()

```
virtual DDSDataReader * DDSReadCondition::get_datareader ( ) [pure virtual]
```

Returns the **DDSDataReader** (p. 1556) associated with the **DDSReadCondition** (p. 1842).

There is exactly one **DDSDataReader** (p. 1556) assicated with each **DDSReadCondition** (p. 1842).

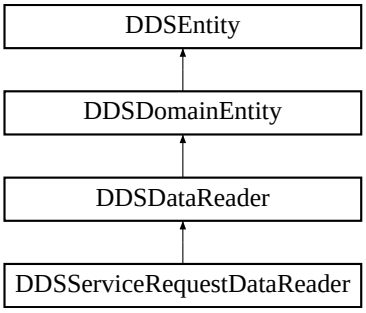
Returns

DDSDataReader (p. 1556) associated with the **DDSReadCondition** (p. 1842).

9.283 DDSServiceRequestDataReader Class Reference

Instantiates `DataReader` < **DDS_ServiceRequest** (p. 1359) > .

Inheritance diagram for DDSServiceRequestDataReader:



Additional Inherited Members

9.283.1 Detailed Description

Instantiates `DataReader` < **DDS_ServiceRequest** (p. 1359) > .

DDSDataReader (p. 1556) of topic **DDS_SERVICE_REQUEST_TOPIC_NAME** (p. 499) used for accessing **DDS_↩ ServiceRequest** (p. 1359) samples.

Instantiates:

<<***generic***>> *(p. 433)* **FooDataReader** (p. 1917)

See also

DDS_ServiceRequest (p. 1359)

DDS_SERVICE_REQUEST_TOPIC_NAME (p. 499)

9.284 DDSServiceRequestTypeSupport Class Reference

Instantiates `TypeSupport` < **DDS_ServiceRequest** (p. 1359) > .

9.284.1 Detailed Description

Instantiates `TypeSupport` < **DDS_ServiceRequest** (p. 1359) > .

Instantiates:

<<***generic***>> *(p. 433)* **FooTypeSupport** (p. 1979)

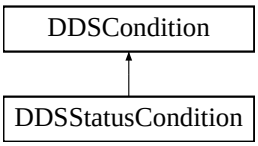
See also

DDS_ServiceRequest (p. 1359)

9.285 DDSStatusCondition Class Reference

<<***interface***>> *(*p. *433)* A specific **DDSCondition** (p. 1544) that is associated with each **DDSEntity** (p. 1731).

Inheritance diagram for DDSStatusCondition:



Public Member Functions

- virtual **DDS_StatusMask get_enabled_statuses** ()=0

  Get the list of statuses enabled on an ***DDSEntity*** *(*p. *1731).*
- virtual **DDS_ReturnCode_t set_enabled_statuses** (**DDS_StatusMask** mask)=0

  This operation defines the list of communication statuses that determine the `trigger_value` *of the* ***DDSStatus↩ Condition*** *(*p. *1846).*
- virtual **DDSEntity** ∗ **get_entity** ()=0

  Get the ***DDSEntity*** *(*p. *1731) associated with the* ***DDSStatusCondition*** *(*p. *1846).*

9.285.1 Detailed Description

<<***interface***>> *(*p. *433)* A specific **DDSCondition** (p. 1544) that is associated with each **DDSEntity** (p. 1731).

The `trigger_value` of the **DDSStatusCondition** (p. 1846) depends on the communication status of that entity (e.↩ g., arrival of data, loss of information, etc.), 'filtered' by the set of `enabled_statuses` on the **DDSStatusCondition** (p. 1846).

See also

Status Kinds (p. 533)

DDSWaitSet (p. 1897), **DDSCondition** (p. 1544)

DDSListener (p. 1794)

9.285.2 Member Function Documentation

9.285.2.1 get_enabled_statuses()

```
virtual DDS_StatusMask DDSStatusCondition::get_enabled_statuses ( ) [pure virtual]
```

Get the list of statuses enabled on an **DDSEntity** (p. 1731).

Returns

list of enabled statuses.

9.285.2.2 set_enabled_statuses()

```
virtual DDS_ReturnCode_t DDSStatusCondition::set_enabled_statuses (
            DDS_StatusMask mask ) [pure virtual]
```

This operation defines the list of communication statuses that determine the `trigger_value` of the **DDSStatus↩ Condition** (p. 1846).

This operation may change the `trigger_value` of the **DDSStatusCondition** (p. 1846).

DDSWaitSet (p. 1897) objects' behavior depends on the changes of the `trigger_value` of their attached conditions. Therefore, any **DDSWaitSet** (p. 1897) to which the **DDSStatusCondition** (p. 1846) is attached is potentially affected by this operation.

If this function is not invoked, the default list of enabled statuses includes all the statuses.

Parameters

mask	<<***in***>> *(*p. *433)* the list of enables statuses (see **Status Kinds** (p. 533))

Exceptions

One	of the **Standard Return Codes** (p. 532)

9.285.2.3 get_entity()

```
virtual DDSEntity * DDSStatusCondition::get_entity ( ) [pure virtual]
```

Get the **DDSEntity** (p. 1731) associated with the **DDSStatusCondition** (p. 1846).

There is exactly one **DDSEntity** (p. 1731) assocated with each **DDSStatusCondition** (p. 1846).

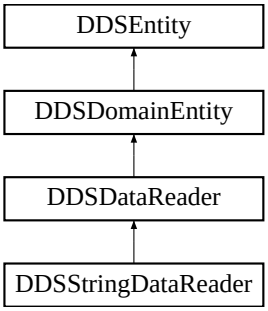
Returns

DDSEntity (p. 1731) associated with the **DDSStatusCondition** (p. 1846).

9.286 DDSStringDataReader Class Reference

<<***interface***>> *(*p. *433)* Instantiates `DataReader` < char* >.

Inheritance diagram for DDSStringDataReader:



Public Member Functions

- virtual **DDS_ReturnCode_t read** (**DDS_StringSeq** &received_data, **DDS_SampleInfoSeq** &info_seq, **DDS_Long** max_samples= **DDS_LENGTH_UNLIMITED**, **DDS_SampleStateMask** sample_states= **DDS_ANY_SAMPLE_STATE**, **DDS_ViewStateMask** view_states= **DDS_ANY_VIEW_STATE**, **DDS_InstanceStateMask** instance_states= **DDS_ANY_INSTANCE_STATE**)

  *Access a collection of data samples from the **DDSDataReader** (*p. *1556).*
- virtual **DDS_ReturnCode_t take** (**DDS_StringSeq** &received_data, **DDS_SampleInfoSeq** &info_seq, **DDS_Long** max_samples= **DDS_LENGTH_UNLIMITED**, **DDS_SampleStateMask** sample_states= **DDS_ANY_SAMPLE_STATE**, **DDS_ViewStateMask** view_states= **DDS_ANY_VIEW_STATE**, **DDS_InstanceStateMask** instance_states= **DDS_ANY_INSTANCE_STATE**)

  *Access a collection of data-samples from the **DDSDataReader** (*p. *1556).*
- virtual **DDS_ReturnCode_t read_w_condition** (**DDS_StringSeq** &received_data, **DDS_SampleInfoSeq** &info_seq, **DDS_Long** max_samples, **DDSReadCondition** *condition)

  *Accesses via **DDSStringDataReader::read** (*p. *1849) the samples that match the criteria specified in the **DDSReadCondition** (*p. *1842).*
- virtual **DDS_ReturnCode_t take_w_condition** (**DDS_StringSeq** &received_data, **DDS_SampleInfoSeq** &info_seq, **DDS_Long** max_samples, **DDSReadCondition** *condition)

  *Analogous to **DDSStringDataReader::read_w_condition** (*p. *1849) except it accesses samples via the **DDSStringDataReader::take** (*p. *1849) operation.*
- virtual **DDS_ReturnCode_t read_next_sample** (char *received_data, **DDS_SampleInfo** &sample_info)

  *Copies the next not-previously-accessed data value from the **DDSDataReader** (*p. *1556).*
- virtual **DDS_ReturnCode_t take_next_sample** (char *received_data, **DDS_SampleInfo** &sample_info)

  *Copies the next not-previously-accessed data value from the **DDSDataReader** (*p. *1556).*
- virtual **DDS_ReturnCode_t return_loan** (**DDS_StringSeq** &received_data, **DDS_SampleInfoSeq** &info_seq)

  *Indicates to the **DDSDataReader** (*p. *1556) that the application is done accessing the collection of `received_data` and `info_seq` obtained by some earlier invocation of read or take on the **DDSDataReader** (*p. *1556).*

Static Public Member Functions

- static **DDSStringDataReader** * **narrow** (**DDSDataReader** *reader)

  *Narrow the given **DDSDataReader** (*p. *1556) pointer to a **DDSStringDataReader** (*p. *1848) pointer.*

9.286.1 Detailed Description

<<***interface***>> *(p. 433)* Instantiates `DataReader` < char* >.

See also

FooDataReader (p. 1917)

DDSDataReader (p. 1556)

String Support (p. 757)

9.286.2 Member Function Documentation

9.286.2.1 read()

```
virtual  DDS_ReturnCode_t DDSStringDataReader::read (
              DDS_StringSeq & received_data,
              DDS_SampleInfoSeq & info_seq,
              DDS_Long max_samples =  DDS_LENGTH_UNLIMITED,
              DDS_SampleStateMask sample_states =  DDS_ANY_SAMPLE_STATE,
              DDS_ViewStateMask view_states =  DDS_ANY_VIEW_STATE,
              DDS_InstanceStateMask instance_states =  DDS_ANY_INSTANCE_STATE )  [virtual]
```

Access a collection of data samples from the **DDSDataReader** (p. 1556).

See also

FooDataReader::read (p. 1919)

9.286.2.2 take()

```
virtual  DDS_ReturnCode_t DDSStringDataReader::take (
              DDS_StringSeq & received_data,
              DDS_SampleInfoSeq & info_seq,
              DDS_Long max_samples =  DDS_LENGTH_UNLIMITED,
              DDS_SampleStateMask sample_states =  DDS_ANY_SAMPLE_STATE,
              DDS_ViewStateMask view_states =  DDS_ANY_VIEW_STATE,
              DDS_InstanceStateMask instance_states =  DDS_ANY_INSTANCE_STATE )  [virtual]
```

Access a collection of data-samples from the **DDSDataReader** (p. 1556).

See also

FooDataReader::take (p. 1920)

9.286.2.3 read_w_condition()

```
virtual DDS_ReturnCode_t DDSStringDataReader::read_w_condition (
             DDS_StringSeq & received_data,
             DDS_SampleInfoSeq & info_seq,
             DDS_Long max_samples,
             DDSReadCondition * condition ) [virtual]
```

Accesses via **DDSStringDataReader::read** (p. 1849) the samples that match the criteria specified in the **DDSReadCondition** (p. 1842).

See also

FooDataReader::read_w_condition (p. 1925)

9.286.2.4 take_w_condition()

```
virtual DDS_ReturnCode_t DDSStringDataReader::take_w_condition (
             DDS_StringSeq & received_data,
             DDS_SampleInfoSeq & info_seq,
             DDS_Long max_samples,
             DDSReadCondition * condition ) [virtual]
```

Analogous to **DDSStringDataReader::read_w_condition** (p. 1849) except it accesses samples via the **DDSStringDataReader::take** (p. 1849) operation.

See also

FooDataReader::take_w_condition (p. 1926)

9.286.2.5 read_next_sample()

```
virtual DDS_ReturnCode_t DDSStringDataReader::read_next_sample (
            char * received_data,
             DDS_SampleInfo & sample_info ) [virtual]
```

Copies the next not-previously-accessed data value from the **DDSDataReader** (p. 1556).

See also

FooDataReader::read_next_sample (p. 1927)

9.286.2.6 take_next_sample()

```
virtual  DDS_ReturnCode_t DDSStringDataReader::take_next_sample (
             char * received_data,
              DDS_SampleInfo & sample_info )  [virtual]
```

Copies the next not-previously-accessed data value from the **DDSDataReader** (p. 1556).

See also

FooDataReader::take_next_sample (p. 1928)

9.286.2.7 return_loan()

```
virtual  DDS_ReturnCode_t DDSStringDataReader::return_loan (
              DDS_StringSeq & received_data,
              DDS_SampleInfoSeq & info_seq )  [virtual]
```

Indicates to the **DDSDataReader** (p. 1556) that the application is done accessing the collection of `received_data` and `info_seq` obtained by some earlier invocation of read or take on the **DDSDataReader** (p. 1556).

See also

FooDataReader::return_loan (p. 1940)

9.286.2.8 narrow()

```
static  DDSStringDataReader * DDSStringDataReader::narrow (
              DDSDataReader * reader )  [static]
```

Narrow the given **DDSDataReader** (p. 1556) pointer to a **DDSStringDataReader** (p. 1848) pointer.

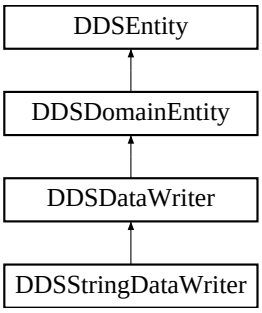
See also

FooDataReader::narrow (p. 1919)

9.287 DDSStringDataWriter Class Reference

<<***interface***>> *(*p. *433)* Instantiates `DataWriter` < char* >.

Inheritance diagram for DDSStringDataWriter:



Public Member Functions

- virtual char * **create_data** (const **DDS_TypeAllocationParams_t** &alloc_params)

  Creates a string data instance.
- virtual **DDS_Boolean delete_data** (char *sample, const **DDS_TypeDeallocationParams_t** &dealloc_params)

  Destroys a string data instance created by ***DDSStringDataWriter::create_data*** *(*p. *1853).*
- virtual **DDS_ReturnCode_t write** (const char *instance_data, const **DDS_InstanceHandle_t** &handle)

  Modifies the value of a string data instance.
- virtual **DDS_ReturnCode_t write_w_timestamp** (const char *instance_data, const **DDS_InstanceHandle_t** &handle, const **DDS_Time_t** &source_timestamp)

  Performs the same function as ***DDSStringDataWriter::write*** *(*p. *1854) except that it also provides the value for the* `source_timestamp`.
- virtual **DDS_ReturnCode_t write_w_params** (const char *instance_data, **DDS_WriteParams_t** &params)

  Performs the same function as ***DDSStringDataWriter::write*** *(*p. *1854) except that it also allows specification of the instance handle, source timestamp, publication priority, and cookie.*

Static Public Member Functions

- static **DDSStringDataWriter** * **narrow** (**DDSDataWriter** *writer)

  Narrow the given ***DDSDataWriter*** *(*p. *1589) pointer to a* ***DDSStringDataWriter*** *(*p. *1852) pointer.*

9.287.1 Detailed Description

<<***interface***>> *(*p. *433)* Instantiates `DataWriter` < char* >.

See also

FooDataWriter (p. 1944)

DDSDataWriter (p. 1589)

String Support (p. 757)

9.287.2 Member Function Documentation

9.287.2.1 narrow()

```
static DDSStringDataWriter * DDSStringDataWriter::narrow (
            DDSDataWriter * writer ) [static]
```

Narrow the given **DDSDataWriter** (p. 1589) pointer to a **DDSStringDataWriter** (p. 1852) pointer.

See also

FooDataWriter::narrow (p. 1946)

9.287.2.2 create_data()

```
virtual char * DDSStringDataWriter::create_data (
            const DDS_TypeAllocationParams_t & alloc_params ) [virtual]
```

Creates a string data instance.

The size of the instance including the NULL terminated character is determined by the DataWriter property **dds.builtin_type.string.alloc_size**.

Default size: **dds.builtin_type.string.max_size** property of DomainParticipant if defined. Otherwise 1024.

Created instances must be deleted with **DDSStringDataWriter::delete_data** (p. 1853).

Returns

Newly created string data, or NULL on failure.

See also

BuiltinTypeMemoryManagement section of Built-in Types

9.287.2.3 delete_data()

```
virtual  DDS_Boolean DDSStringDataWriter::delete_data (
             char * sample,
             const  DDS_TypeDeallocationParams_t & dealloc_params )  [virtual]
```

Destroys a string data instance created by **DDSStringDataWriter::create_data** (p. 1853).

Returns

DDS_BOOLEAN_TRUE (p. 513) upon successful deletion.

9.287.2.4 write()

```
virtual  DDS_ReturnCode_t DDSStringDataWriter::write (
             const char * instance_data,
             const  DDS_InstanceHandle_t & handle )  [virtual]
```

Modifies the value of a string data instance.

See also

FooDataWriter::write (p. 1951)

9.287.2.5 write_w_timestamp()

```
virtual  DDS_ReturnCode_t DDSStringDataWriter::write_w_timestamp (
             const char * instance_data,
             const  DDS_InstanceHandle_t & handle,
             const  DDS_Time_t & source_timestamp )  [virtual]
```

Performs the same function as **DDSStringDataWriter::write** (p. 1854) except that it also provides the value for the `source_timestamp`.

See also

FooDataWriter::write_w_timestamp (p. 1955)

9.287.2.6 write_w_params()

```
virtual DDS_ReturnCode_t DDSStringDataWriter::write_w_params (
            const char * instance_data,
             DDS_WriteParams_t & params ) [virtual]
```

Performs the same function as **DDSStringDataWriter::write** (p. 1854) except that it also allows specification of the instance handle, source timestamp, publication priority, and cookie.

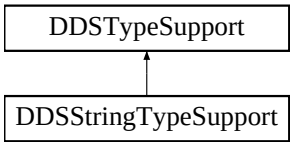
See also

FooDataWriter::write_w_params (p. 1956)

9.288 DDSStringTypeSupport Class Reference

<<***interface***>> *(p. 433)* String type support.

Inheritance diagram for DDSStringTypeSupport:



Static Public Member Functions

- static **DDS_ReturnCode_t register_type** (**DDSDomainParticipant** *participant, const char *type_name="DDS::String")

  Allows an application to communicate to RTI Connext the existence of the char data type.*
- static **DDS_ReturnCode_t unregister_type** (**DDSDomainParticipant** *participant, const char *type_name="DDS::String")

  Allows an application to unregister the char data type from RTI Connext. After calling unregister_type, no further communication using this type is possible.*
- static const char * **get_type_name** ()

  Get the default name for the char type.*
- static void **print_data** (const char *a_data)

  <<***extension***>> *(p. 432) Print value of data type to standard out.*
- static **DDS_TypeCode** * **get_typecode** ()

  <<***extension***>> *(p. 432) Retrieves the TypeCode for the Type.*
- static **DDS_ReturnCode_t serialize_data_to_cdr_buffer** (char *buffer, unsigned int &length, const char *a_data)

  <<***extension***>> *(p. 432) Serializes the input sample into a CDR buffer of octets.*
- static **DDS_ReturnCode_t serialize_data_to_cdr_buffer_ex** (char *buffer, unsigned int &length, const char *a_data, **DDS_DataRepresentationId_t** representation)

  <<***extension***>> *(p. 432) Serializes the input sample into a buffer of octets.*
- static **DDS_ReturnCode_t deserialize_data_from_cdr_buffer** (char **a_data, const char *buffer, unsigned int length)

  <<***extension***>> *(p. 432) Deserializes a sample from a buffer of octets.*
- static **DDS_ReturnCode_t data_to_string** (char *sample, char *str, **DDS_UnsignedLong** &str_size, const **DDS_PrintFormatProperty** &property)

  <<***extension***>> *(p. 432) Get the string representation of an input sample.*

9.288.1 Detailed Description

<<***interface***>> *(p. 433)* String type support.

9.288.2 Member Function Documentation

9.288.2.1 register_type()

```
static DDS_ReturnCode_t DDSStringTypeSupport::register_type (
            DDSDomainParticipant * participant,
           const char * type_name = "DDS::String" )  [static]
```

Allows an application to communicate to RTI Connext the existence of the char* data type.

By default, The char* built-in type is automatically registered when a DomainParticipant is created using the type_name returned by **DDSStringTypeSupport::get_type_name** (p. 1857). Therefore, the usage of this function is optional and it is only required when the automatic built-in type registration is disabled using the participant property "dds.builtin_↩ type.auto_register".

This method can also be used to register the same **DDSStringTypeSupport** (p. 1855) with a **DDSDomainParticipant** (p. 1619) using different values for the type_name.

If `register_type` is called multiple times with the same **DDSDomainParticipant** (p. 1619) and `type_name`, the second (and subsequent) registrations are ignored by the operation.

Parameters

participant	<<***in***>> *(p. 433)* the **DDSDomainParticipant** (p. 1619) to register the data type char* with. Cannot be NULL.
type_name	<<***in***>> *(p. 433)* the type name under with the data type char* is registered with the participant; this type name is used when creating a new **DDSTopic** (p. 1885). (See **DDSDomainParticipant::create_topic** (p. 1650).) The name may not be NULL or longer than 255 characters.

Exceptions

One	of the **Standard Return Codes** (p. 532), **DDS_RETCODE_PRECONDITION_NOT_MET** (p. 532) or **DDS_RETCODE_OUT_OF_RESOURCES** (p. 533).

MT Safety:

UNSAFE on the FIRST call. It is not safe for two threads to simultaneously make the first call to register a type. Subsequent calls are thread safe.

See also

DDSDomainParticipant::create_topic (p. 1650)

9.288.2.2 unregister_type()

```
static DDS_ReturnCode_t DDSStringTypeSupport::unregister_type (
            DDSDomainParticipant * participant,
           const char * type_name = "DDS::String" ) [static]
```

Allows an application to unregister the char* data type from RTI Connext. After calling unregister_type, no further communication using this type is possible.

Precondition

The char* type with `type_name` is registered with the participant and all **DDSTopic** (p. 1885) objects referencing the type have been destroyed. If the type is not registered with the participant, or if any **DDSTopic** (p. 1885) is associated with the type, the operation will fail with **DDS_RETCODE_ERROR** (p. 532).

Postcondition

All information about the type is removed from RTI Connext. No further communication using this type is possible.

Parameters

participant	<<***in***>> *(*p. *433)* the **DDSDomainParticipant** (p. 1619) to unregister the data type char* from. Cannot be NULL.
type_name	<<***in***>> *(*p. *433)* the type name under with the data type char* is registered with the participant. The name should match a name that has been previously used to register a type with the participant. Cannot be NULL.

Exceptions

One	of the **Standard Return Codes** (p. 532), **DDS_RETCODE_BAD_PARAMETER** (p. 532) or **DDS_RETCODE_ERROR** (p. 532)

MT Safety:

SAFE.

See also

DDSStringTypeSupport::register_type (p. 1856)

9.288.2.3 get_type_name()

```
static const char * DDSStringTypeSupport::get_type_name ( )  [static]
```

Get the default name for the char* type.

Can be used for calling **DDSStringTypeSupport::register_type** (p. 1856) or creating **DDSTopic** (p. 1885).

Returns

default name for the char* type.

See also

DDSStringTypeSupport::register_type (p. 1856)

DDSDomainParticipant::create_topic (p. 1650)

9.288.2.4 print_data()

```
static void DDSStringTypeSupport::print_data (
            const char * a_data )  [static]
```

<<***extension***>> *(p. 432)* Print value of data type to standard out.

The *generated* implementation of the operation knows how to print value of a data type.

Parameters

a_data	<<***in***>> *(p. 433)* String to be printed.

9.288.2.5 get_typecode()

```
static  DDS_TypeCode * DDSStringTypeSupport::get_typecode ( )  [static]
```

<<***extension***>> *(p. 432)* Retrieves the TypeCode for the Type.

See also

FooTypeSupport::get_typecode (p. 1991)

9.288.2.6 serialize_data_to_cdr_buffer()

```
static DDS_ReturnCode_t DDSStringTypeSupport::serialize_data_to_cdr_buffer (
            char * buffer,
            unsigned int & length,
            const char * a_data )  [static]
```

<<***extension***>> *(p. 432)* Serializes the input sample into a CDR buffer of octets.

See also

FooTypeSupport::serialize_data_to_cdr_buffer (p. 1989)

9.288.2.7 serialize_data_to_cdr_buffer_ex()

```
static DDS_ReturnCode_t DDSStringTypeSupport::serialize_data_to_cdr_buffer_ex (
            char * buffer,
            unsigned int & length,
            const char * a_data,
             DDS_DataRepresentationId_t representation )  [static]
```

<<***extension***>> *(p. 432)* Serializes the input sample into a buffer of octets.

See also

FooTypeSupport::serialize_data_to_cdr_buffer (p. 1989)

9.288.2.8 deserialize_data_from_cdr_buffer()

```
static DDS_ReturnCode_t DDSStringTypeSupport::deserialize_data_from_cdr_buffer (
            char ** a_data,
            const char * buffer,
            unsigned int length )  [static]
```

<<***extension***>> *(p. 432)* Deserializes a sample from a buffer of octets.

See also

FooTypeSupport::deserialize_data_from_cdr_buffer (p. 1990)

9.288.2.9 data_to_string()

```
static  DDS_ReturnCode_t DDSStringTypeSupport::data_to_string (
             char * sample,
             char * str,
              DDS_UnsignedLong & str_size,
             const  DDS_PrintFormatProperty & property )  [static]
```

<<***extension***>> *(*p. *432)* Get the string representation of an input sample.

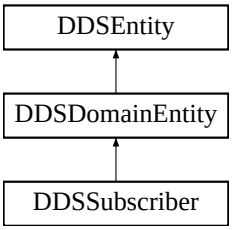
See also

FooTypeSupport::data_to_string (p. 1991)

9.289 DDSSubscriber Class Reference

<<***interface***>> *(*p. *433)* A subscriber is the object responsible for actually receiving data from a subscription.

Inheritance diagram for DDSSubscriber:



Public Member Functions

- virtual **DDS_ReturnCode_t get_default_datareader_qos** (**DDS_DataReaderQos** &qos)=0

  *Copies the default **DDS_DataReaderQos** (*p. *858) values into the provided **DDS_DataReaderQos** (*p. *858) instance.*
- virtual **DDS_ReturnCode_t set_default_datareader_qos** (const **DDS_DataReaderQos** &qos)=0

  *Sets the default **DDS_DataReaderQos** (*p. *858) values for this subscriber.*
- virtual **DDS_ReturnCode_t set_default_datareader_qos_with_profile** (const char ∗library_name, const char ∗profile_name)=0

  <<***extension***>> *(*p. *432) Set the default **DDS_DataReaderQos** (*p. *858) values for this subscriber based on the input XML QoS profile.*
- virtual **DDS_ReturnCode_t set_default_library** (const char ∗library_name)=0

  <<***extension***>> *(*p. *432) Sets the default XML library for a **DDSSubscriber** (*p. *1860).*
- virtual const char ∗ **get_default_library** ()=0

  <<***extension***>> *(*p. *432) Gets the default XML library associated with a **DDSSubscriber** (*p. *1860).*
- virtual **DDS_ReturnCode_t set_default_profile** (const char ∗library_name, const char ∗profile_name)=0

  <<***extension***>> *(*p. *432) Sets the default XML profile for a **DDSSubscriber** (*p. *1860).*
- virtual const char ∗ **get_default_profile** ()=0

  <<***extension***>> *(*p. *432) Gets the default XML profile associated with a **DDSSubscriber** (*p. *1860).*

- virtual const char * **get_default_profile_library** ()=0

  *<<**extension**>> (p. 432) Gets the library where the default XML QoS profile is contained for a **DDSSubscriber** (p. 1860).*
- virtual **DDSDataReader** * **create_datareader** (**DDSTopicDescription** *topic, const **DDS_DataReaderQos** &qos, **DDSDataReaderListener** *listener, **DDS_StatusMask** mask)=0

  *Creates a **DDSDataReader** (p. 1556) that will be attached and belong to the **DDSSubscriber** (p. 1860).*
- virtual **DDSDataReader** * **create_datareader_with_profile** (**DDSTopicDescription** *topic, const char *library_name, const char *profile_name, **DDSDataReaderListener** *listener, **DDS_StatusMask** mask)=0

  *<<**extension**>> (p. 432) Creates a **DDSDataReader** (p. 1556) object using the **DDS_DataReaderQos** (p. 858) associated with the input XML QoS profile.*
- virtual **DDS_ReturnCode_t** **delete_datareader** (**DDSDataReader** *a_datareader)=0

  *Deletes a **DDSDataReader** (p. 1556) that belongs to the **DDSSubscriber** (p. 1860).*
- virtual **DDS_ReturnCode_t** **delete_contained_entities** ()=0

  *Deletes all the entities that were created by means of the "create" operation on the **DDSSubscriber** (p. 1860).*
- virtual **DDSDataReader** * **lookup_datareader** (const char *topic_name)=0

  *Retrieves an existing **DDSDataReader** (p. 1556).*
- virtual **DDS_ReturnCode_t** **begin_access** ()=0

  *Indicates that the application is about to access the data samples in any of the **DDSDataReader** (p. 1556) objects attached to the **DDSSubscriber** (p. 1860).*
- virtual **DDS_ReturnCode_t** **end_access** ()=0

  *Indicates that the application has finished accessing the data samples in **DDSDataReader** (p. 1556) objects managed by the **DDSSubscriber** (p. 1860).*
- virtual **DDS_ReturnCode_t** **get_datareaders** (**DDSDataReaderSeq** &readers, **DDS_SampleStateMask** sample_states, **DDS_ViewStateMask** view_states, **DDS_InstanceStateMask** instance_states)=0

  *Allows the application to access the **DDSDataReader** (p. 1556) objects that contain samples with the specified `sample_states`, `view_states` and `instance_states`.*
- virtual **DDS_ReturnCode_t** **get_all_datareaders** (**DDSDataReaderSeq** &readers)=0

  Retrieve all the DataReaders created from this Subscriber.
- virtual **DDS_ReturnCode_t** **notify_datareaders** ()=0

  *Invokes the operation **DDSDataReaderListener::on_data_available()** (p. 1585) on the **DDSDataReaderListener** (p. 1583) objects attached to contained **DDSDataReader** (p. 1556) entities with **DDS_DATA_AVAILABLE_STATUS** (p. 541) that is considered changed as described in **Changes in read communication status** (p. 536).*
- virtual **DDSDomainParticipant** * **get_participant** ()=0

  *Returns the **DDSDomainParticipant** (p. 1619) to which the **DDSSubscriber** (p. 1860) belongs.*
- virtual **DDS_ReturnCode_t** **copy_from_topic_qos** (**DDS_DataReaderQos** &datareader_qos, const **DDS_TopicQos** &topic_qos)=0

  *Copies the policies in the **DDS_TopicQos** (p. 1397) to the corresponding policies in the **DDS_DataReaderQos** (p. 858).*
- virtual **DDS_ReturnCode_t** **set_qos** (const **DDS_SubscriberQos** &qos)=0

  Sets the subscriber QoS.
- virtual **DDS_ReturnCode_t** **set_qos_with_profile** (const char *library_name, const char *profile_name)=0

  *<<**extension**>> (p. 432) Change the QoS of this subscriber using the input XML QoS profile.*
- virtual **DDS_ReturnCode_t** **get_qos** (**DDS_SubscriberQos** &qos)=0

  Gets the subscriber QoS.
- virtual **DDS_ReturnCode_t** **set_listener** (**DDSSubscriberListener** *l, **DDS_StatusMask** mask= **DDS_STATUS_MASK_ALL**)=0

  Sets the subscriber listener.
- virtual **DDSSubscriberListener** * **get_listener** ()=0

  Get the subscriber listener.
- virtual **DDSDataReader** * **lookup_datareader_by_name** (const char *datareader_name)=0

  *<<**extension**>> (p. 432) Retrieves a **DDSDataReader** (p. 1556) contained within the **DDSSubscriber** (p. 1860) the **DDSDataReader** (p. 1556) entity name.*

9.289.1 Detailed Description

<<***interface***>> *(*p. *433)* A subscriber is the object responsible for actually receiving data from a subscription.

QoS:

DDS_SubscriberQos (p. 1367)

Status:

DDS_DATA_ON_READERS_STATUS (p. 541)

Listener:

DDSSubscriberListener (p. 1881)

A subscriber acts on the behalf of one or several **DDSDataReader** (p. 1556) objects that are related to it. When it receives data (from the other parts of the system), it builds the list of concerned **DDSDataReader** (p. 1556) objects and then indicates to the application that data is available through its listener or by enabling related conditions.

The application can access the list of concerned **DDSDataReader** (p. 1556) objects through the operation **DDSSubscriber::get_datareaders** (p. 1874) and then access the data available through operations on the **DDSData↩ Reader** (p. 1556).

The following operations may be called even if the **DDSSubscriber** (p. 1860) is not enabled. Other operations will the value **DDS_RETCODE_NOT_ENABLED** (p. 533) if called on a disabled **DDSSubscriber** (p. 1860):

- **DDSEntity::enable** (p. 1734),
- **DDSSubscriber::set_qos** (p. 1877), **DDSSubscriber::get_qos** (p. 1879) , idref_Subscriber_set_qos_with_↩ profile
- **DDSSubscriber::set_listener** (p. 1879), **DDSSubscriber::get_listener** (p. 1880),
- **DDSEntity::get_statuscondition** (p. 1735), **DDSEntity::get_status_changes** (p. 1735)
- **DDSSubscriber::create_datareader** (p. 1867), **DDSSubscriber::create_datareader_with_profile** (p. 1869), **DDSSubscriber::delete_contained_entities** (p. 1871), **DDSSubscriber::delete_datareader** (p. 1870),
- **DDSSubscriber::set_default_datareader_qos** (p. 1863), **DDSSubscriber::set_default_datareader_qos↩ _with_profile** (p. 1864), **DDSSubscriber::get_default_datareader_qos** (p. 1863), **DDSSubscriber::set_↩ default_library** (p. 1865), **DDSSubscriber::set_default_profile** (p. 1866)

All operations except for **DDSSubscriber::set_qos** (p. 1877), **DDSSubscriber::set_qos_with_profile** (p. 1878), **DDSSubscriber::get_qos** (p. 1879), **DDSSubscriber::set_listener** (p. 1879), **DDSSubscriber::get_listener** (p. 1880), **DDSEntity::enable** (p. 1734) and **DDSSubscriber::create_datareader** (p. 1867) may fail with **DDS_↩ RETCODE_NOT_ENABLED** (p. 533).

See also

Operations Allowed in Listener Callbacks (p. 1796)

Examples

HelloWorld_subscriber.cxx.

9.289.2 Member Function Documentation

9.289.2.1 get_default_datareader_qos()

```
virtual DDS_ReturnCode_t DDSSubscriber::get_default_datareader_qos (
            DDS_DataReaderQos & qos ) [pure virtual]
```

Copies the default **DDS_DataReaderQos** (p. 858) values into the provided **DDS_DataReaderQos** (p. 858) instance.

The retrieved `qos` will match the set of values specified on the last successful call to **DDSSubscriber::set_default_datareader_qos** (p. 1863), or **DDSSubscriber::set_default_datareader_qos_with_profile** (p. 1864), or else, if the call was never made, the default values from is owning **DDSDomainParticipant** (p. 1619).

This method may potentially allocate memory depending on the sequences contained in some QoS policies.

MT Safety:

UNSAFE. It is not safe to retrieve the default QoS value from a subscriber while another thread may be simultaneously calling **DDSSubscriber::set_default_datareader_qos** (p. 1863)

Parameters

qos	<<***inout***>> *(p. 433)* **DDS_DataReaderQos** (p. 858) to be filled-up.

Exceptions

One	of the **Standard Return Codes** (p. 532)

See also

DDS_DATAREADER_QOS_DEFAULT (p. 136)

DDSSubscriber::create_datareader (p. 1867)

9.289.2.2 set_default_datareader_qos()

```
virtual DDS_ReturnCode_t DDSSubscriber::set_default_datareader_qos (
            const DDS_DataReaderQos & qos ) [pure virtual]
```

Sets the default **DDS_DataReaderQos** (p. 858) values for this subscriber.

This call causes the default values inherited from the owning **DDSDomainParticipant** (p. 1619) to be overridden.

This default value will be used for newly created **DDSDataReader** (p. 1556) if **DDS_DATAREADER_QOS_DEFAULT** (p. 136) is specified as the `qos` parameter when **DDSSubscriber::create_datareader** (p. 1867) is called.

Precondition

The specified QoS policies must be consistent, or else the operation will have no effect and fail with **DDS_↩ RETCODE_INCONSISTENT_POLICY** (p. 533)

MT Safety:

UNSAFE. It is not safe to set the default QoS value from a subscriber while another thread may be simultaneously calling **DDSSubscriber::set_default_datareader_qos** (p. 1863), **DDSSubscriber::get_default↩ _datareader_qos** (p. 1863) or calling **DDSSubscriber::create_datareader** (p. 1867) with **DDS_DATAREADER↩ _QOS_DEFAULT** (p. 136) as the `qos` parameter.

Parameters

qos	*<<**in**>> (p. 433)* The default **DDS_DataReaderQos** (p. 858) to be set to. The special value **DDS_DATAREADER_QOS_DEFAULT** (p. 136) may be passed as `qos` to indicate that the default QoS should be reset back to the initial values the factory would used if **DDSSubscriber::set_default_datareader_qos** (p. 1863) had never been called.

Exceptions

One	of the **Standard Return Codes** (p. 532), or or **DDS_RETCODE_INCONSISTENT_POLICY** (p. 533)

9.289.2.3 set_default_datareader_qos_with_profile()

```
virtual  DDS_ReturnCode_t DDSSubscriber::set_default_datareader_qos_with_profile (
             const char * library_name,
             const char * profile_name )  [pure virtual]
```

*<<**extension**>> (p. 432)* Set the default **DDS_DataReaderQos** (p. 858) values for this subscriber based on the input XML QoS profile.

This default value will be used for newly created **DDSDataReader** (p. 1556) if **DDS_DATAREADER_QOS_DEFAULT** (p. 136) is specified as the `qos` parameter when **DDSSubscriber::create_datareader** (p. 1867) is called.

Precondition

The **DDS_DataReaderQos** (p. 858) contained in the specified XML QoS profile must be consistent, or else the operation will have no effect and fail with **DDS_RETCODE_INCONSISTENT_POLICY** (p. 533)

MT Safety:

UNSAFE. It is not safe to set the default QoS value from a **DDSSubscriber** (p. 1860) while another thread may be simultaneously calling **DDSSubscriber::set_default_datareader_qos** (p. 1863), **DDSSubscriber↩ ::get_default_datareader_qos** (p. 1863) or calling **DDSSubscriber::create_datareader** (p. 1867) with **DDS_↩ DATAREADER_QOS_DEFAULT** (p. 136) as the `qos` parameter.

Parameters

library_name	<<***in***>> *(p. 433)* Library name containing the XML QoS profile. If library_name is null RTI Connext will use the default library (see **DDSSubscriber::set_default_library** (p. 1865)).
profile_name	<<***in***>> *(p. 433)* XML QoS Profile name. If profile_name is null RTI Connext will use the default profile (see **DDSSubscriber::set_default_profile** (p. 1866)).

If the input profile cannot be found the method fails with **DDS_RETCODE_ERROR** (p. 532).

Exceptions

One	of the **Standard Return Codes** (p. 532), or **DDS_RETCODE_INCONSISTENT_POLICY** (p. 533)

See also

DDS_DATAREADER_QOS_DEFAULT (p. 136)

DDSSubscriber::create_datareader_with_profile (p. 1869)

9.289.2.4 set_default_library()

```
virtual  DDS_ReturnCode_t DDSSubscriber::set_default_library (
             const char * library_name )  [pure virtual]
```

<<***extension***>> *(p. 432)* Sets the default XML library for a **DDSSubscriber** (p. 1860).

This method specifies the library that will be used as the default the next time a default library is needed during a call to one of this Subscriber's operations.

Any API requiring a library_name as a parameter can use null to refer to the default library.

If the default library is not set, the **DDSSubscriber** (p. 1860) inherits the default from the **DDSDomainParticipant** (p. 1619) (see **DDSDomainParticipant::set_default_library** (p. 1635)).

Parameters

library_name	<<***in***>> *(p. 433)* Library name. If library_name is null any previous default is unset.

Exceptions

One	of the **Standard Return Codes** (p. 532)

See also

DDSSubscriber::get_default_library (p. 1866)

9.289.2.5 get_default_library()

```
virtual const char * DDSSubscriber::get_default_library ( )  [pure virtual]
```

<<***extension***>> *(p. 432)* Gets the default XML library associated with a **DDSSubscriber** (p. 1860).

Returns

The default library or null if the default library was not set.

See also

DDSSubscriber::set_default_library (p. 1865)

9.289.2.6 set_default_profile()

```
virtual  DDS_ReturnCode_t DDSSubscriber::set_default_profile (
            const char * library_name,
            const char * profile_name )  [pure virtual]
```

<<***extension***>> *(p. 432)* Sets the default XML profile for a **DDSSubscriber** (p. 1860).

This method specifies the profile that will be used as the default the next time a default Subscriber profile is needed during a call to one of this Subscriber's operations. When calling a **DDSSubscriber** (p. 1860) method that requires a `profile_name` parameter, you can use NULL to refer to the default profile. (This same information applies to setting a default library.)

If the default profile is not set, the **DDSSubscriber** (p. 1860) inherits the default from the **DDSDomainParticipant** (p. 1619) (see **DDSDomainParticipant::set_default_profile** (p. 1636)).

This method does not set the default QoS for **DDSDataReader** (p. 1556) objects created by this **DDSSubscriber** (p. 1860); for this functionality, use **DDSSubscriber::set_default_datareader_qos_with_profile** (p. 1864) (you may pass in NULL after having called **set_default_profile()** (p. 1866)).

This method does not set the default QoS for newly created Subscribers; for this functionality, use **DDSDomain↩ Participant::set_default_subscriber_qos_with_profile** (p. 1643).

Parameters

library_name	<<***in***>> *(p. 433)* The library name containing the profile.
profile_name	<<***in***>> *(p. 433)* The profile name. If profile_name is null any previous default is unset.

Exceptions

One	of the **Standard Return Codes** (p. 532)

See also

DDSSubscriber::get_default_profile (p. 1867)

DDSSubscriber::get_default_profile_library (p. 1867)

9.289.2.7 get_default_profile()

```
virtual const char * DDSSubscriber::get_default_profile ( )  [pure virtual]
```

<<***extension***>> *(p. 432)* Gets the default XML profile associated with a **DDSSubscriber** (p. 1860).

Returns

The default profile or null if the default profile was not set.

See also

DDSSubscriber::set_default_profile (p. 1866)

9.289.2.8 get_default_profile_library()

```
virtual const char * DDSSubscriber::get_default_profile_library ( )  [pure virtual]
```

<<***extension***>> *(p. 432)* Gets the library where the default XML QoS profile is contained for a **DDSSubscriber** (p. 1860).

The default profile library is automatically set when **DDSSubscriber::set_default_profile** (p. 1866) is called.

This library can be different than the **DDSSubscriber** (p. 1860) default library (see **DDSSubscriber::get_default_↩ library** (p. 1866)).

Returns

The default profile library or null if the default profile was not set.

See also

DDSSubscriber::set_default_profile (p. 1866)

9.289.2.9 create_datareader()

```
virtual DDSDataReader * DDSSubscriber::create_datareader (
            DDSTopicDescription * topic,
          const DDS_DataReaderQos & qos,
            DDSDataReaderListener * listener,
            DDS_StatusMask mask ) [pure virtual]
```

Creates a **DDSDataReader** (p. 1556) that will be attached and belong to the **DDSSubscriber** (p. 1860).

For each application-defined type **Foo** (p. 1916), there is an implied, auto-generated class **FooDataReader** (p. 1917) (an incarnation of **FooDataReader** (p. 1917)) that extends **DDSDataReader** (p. 1556) and contains the operations to read data of type **Foo** (p. 1916).

Note that a common application pattern to construct the QoS for the **DDSDataReader** (p. 1556) is to:

- Retrieve the QoS policies on the associated **DDSTopic** (p. 1885) by means of the **DDSTopic::get_qos** (p. 1889) operation.
- Retrieve the default **DDSDataReader** (p. 1556) qos by means of the **DDSSubscriber::get_default_datareader_qos** (p. 1863) operation.
- Combine those two QoS policies (for example, using **DDSSubscriber::copy_from_topic_qos** (p. 1876)) and selectively modify policies as desired
- Use the resulting QoS policies to construct the **DDSDataReader** (p. 1556).

When a **DDSDataReader** (p. 1556) is created, only those transports already registered are available to the **DDSDataReader** (p. 1556). See **Built-in Transport Plugins** (p. 182) for details on when a builtin transport is registered.

MT Safety:

UNSAFE. If **DDS_DATAREADER_QOS_DEFAULT** (p. 136) is used for the `qos` parameter, it is not safe to create the datareader while another thread may be simultaneously calling **DDSSubscriber::set_default_datareader_qos** (p. 1863).

Precondition

If subscriber is enabled, the topic must be enabled. If it is not, this operation will fail and no **DDSDataReader** (p. 1556) will be created.

The given **DDSTopicDescription** (p. 1892) must have been created from the same participant as this subscriber. If it was created from a different participant, this method will return NULL.

If `qos` is **DDS_DATAREADER_QOS_USE_TOPIC_QOS** (p. 137), `topic` cannot be **DDSMultiTopic** (p. 1797), or else this method will return NULL.

Parameters

topic	<<***in***>> *(p. 433)* The **DDSTopicDescription** (p. 1892) that the **DDSDataReader** (p. 1556) will be associated with. Cannot be NULL.

Parameters

qos	<<***in***>> *(p. 433)* The qos of the **DDSDataReader** (p. 1556). The special value **DDS_DATAREADER_QOS_DEFAULT** (p. 136) can be used to indicate that the **DDSDataReader** (p. 1556) should be created with the default **DDS_DataReaderQos** (p. 858) set in the **DDSSubscriber** (p. 1860). If **DDSTopicDescription** (p. 1892) is of type **DDSTopic** (p. 1885) or **DDSContentFilteredTopic** (p. 1551), the special value **DDS_DATAREADER_QOS_USE_TOPIC_QOS** (p. 137) can be used to indicate that the **DDSDataReader** (p. 1556) should be created with the combination of the default **DDS_DataReaderQos** (p. 858) set on the **DDSSubscriber** (p. 1860) and the **DDS_TopicQos** (p. 1397) (in the case of a **DDSContentFilteredTopic** (p. 1551), the **DDS_TopicQos** (p. 1397) of the related **DDSTopic** (p. 1885)). if **DDS_DATAREADER_QOS_USE_TOPIC_QOS** (p. 137) is used, `topic` cannot be a **DDSMultiTopic** (p. 1797).
listener	<<***in***>> *(p. 433)* The listener of the **DDSDataReader** (p. 1556).
mask	<<***in***>> *(p. 433)*. Changes of communication status to be invoked on the listener. See **DDS_StatusMask** (p. 537).

Returns

A **DDSDataReader** (p. 1556) of a derived class specific to the data-type associated with the **DDSTopic** (p. 1885) or NULL if an error occurred.

See also

FooDataReader (p. 1917)

Specifying QoS on entities (p. 555) for information on setting QoS before entity creation

DDS_DataReaderQos (p. 858) for rules on consistency among QoS

DDSSubscriber::create_datareader_with_profile (p. 1869)

DDSSubscriber::get_default_datareader_qos (p. 1863)

DDSTopic::set_qos (p. 1887)

DDSSubscriber::copy_from_topic_qos (p. 1876)

DDSDataReader::set_listener (p. 1577)

9.289.2.10 create_datareader_with_profile()

```
virtual DDSDataReader * DDSSubscriber::create_datareader_with_profile (
            DDSTopicDescription * topic,
          const char * library_name,
          const char * profile_name,
            DDSDataReaderListener * listener,
            DDS_StatusMask mask ) [pure virtual]
```

<<***extension***>> *(p. 432)* Creates a **DDSDataReader** (p. 1556) object using the **DDS_DataReaderQos** (p. 858) associated with the input XML QoS profile.

The **DDSDataReader** (p. 1556) will be attached and belong to the **DDSSubscriber** (p. 1860).

For each application-defined type **Foo** (p. 1916), there is an implied, auto-generated class **FooDataReader** (p. 1917) (an incarnation of **FooDataReader** (p. 1917)) that extends **DDSDataReader** (p. 1556) and contains the operations to read data of type **Foo** (p. 1916).

When a **DDSDataReader** (p. 1556) is created, only those transports already registered are available to the **DDSDataReader** (p. 1556). See **Built-in Transport Plugins** (p. 182) for details on when a builtin transport is registered.

Precondition

If subscriber is enabled, the topic must be enabled. If it is not, this operation will fail and no **DDSDataReader** (p. 1556) will be created.

The given **DDSTopicDescription** (p. 1892) must have been created from the same participant as this subscriber. If it was created from a different participant, this method will return NULL.

Parameters

topic	<<***in***>> *(*p. *433)* The **DDSTopicDescription** (p. 1892) that the **DDSDataReader** (p. 1556) will be associated with. Cannot be NULL.
library_name	<<***in***>> *(*p. *433)* Library name containing the XML QoS profile. If library_name is null RTI Connext will use the default library (see **DDSSubscriber::set_default_library** (p. 1865)).
profile_name	<<***in***>> *(*p. *433)* XML QoS Profile name. If profile_name is null RTI Connext will use the default profile (see **DDSSubscriber::set_default_profile** (p. 1866)).
listener	<<***in***>> *(*p. *433)* The listener of the **DDSDataReader** (p. 1556).
mask	<<***in***>> (p. *433).* Changes of communication status to be invoked on the listener. See **DDS_StatusMask** (p. 537).

Returns

A **DDSDataReader** (p. 1556) of a derived class specific to the data-type associated with the **DDSTopic** (p. 1885) or NULL if an error occurred.

See also

FooDataReader (p. 1917)

Specifying QoS on entities (p. 555) for information on setting QoS before entity creation

DDS_DataReaderQos (p. 858) for rules on consistency among QoS

DDS_DATAREADER_QOS_DEFAULT (p. 136)

DDS_DATAREADER_QOS_USE_TOPIC_QOS (p. 137)

DDSSubscriber::create_datareader (p. 1867)

DDSSubscriber::get_default_datareader_qos (p. 1863)

DDSTopic::set_qos (p. 1887)

DDSSubscriber::copy_from_topic_qos (p. 1876)

DDSDataReader::set_listener (p. 1577)

Examples

HelloWorld_subscriber.cxx.

9.289.2.11 delete_datareader()

```
virtual DDS_ReturnCode_t DDSSubscriber::delete_datareader (
            DDSDataReader * a_datareader ) [pure virtual]
```

Deletes a **DDSDataReader** (p. 1556) that belongs to the **DDSSubscriber** (p. 1860).

Precondition

If the **DDSDataReader** (p. 1556) does not belong to the **DDSSubscriber** (p. 1860), or if ther are any existing **DDSReadCondition** (p. 1842) or **DDSQueryCondition** (p. 1841) objects that are attached to the **DDSData↩ Reader** (p. 1556), or if there are outstanding loans on samples (as a result of a call to read(), take(), or one of the variants thereof), the operation fails with the error **DDS_RETCODE_PRECONDITION_NOT_MET** (p. 532).

Postcondition

Listener installed on the **DDSDataReader** (p. 1556) will not be called after this method completes successfully.

MT Safety:

UNSAFE. It is not safe to delete an entity while another thread may be simultaneously calling an API that uses the entity.

Parameters

a_datareader	<<***in***>> *(p. 433)* The **DDSDataReader** (p. 1556) to be deleted.

Exceptions

One	of the **Standard Return Codes** (p. 532) or **DDS_RETCODE_PRECONDITION_NOT_MET** (p. 532).

9.289.2.12 delete_contained_entities()

```
virtual DDS_ReturnCode_t DDSSubscriber::delete_contained_entities ( ) [pure virtual]
```

Deletes all the entities that were created by means of the "create" operation on the **DDSSubscriber** (p. 1860).

Deletes all contained **DDSDataReader** (p. 1556) objects. This pattern is applied recursively. In this manner, the operation **DDSSubscriber::delete_contained_entities** (p. 1871) on the **DDSSubscriber** (p. 1860) will end up deleting all the entities recursively contained in the **DDSSubscriber** (p. 1860), including the **DDSQueryCondition** (p. 1841), **DDSReadCondition** (p. 1842), and **DDSTopicQuery** (p. 1895) objects belonging to the contained **DDSDataReader** (p. 1556).

The operation will fail with **DDS_RETCODE_PRECONDITION_NOT_MET** (p. 532) if any of the contained entities is in a state where it cannot be deleted. This will occur, for example, if a contained **DDSDataReader** (p. 1556) cannot

be deleted because the application has called a **FooDataReader::read** (p. 1919) or **FooDataReader::take** (p. 1920) operation and has not called the corresponding **FooDataReader::return_loan** (p. 1940) operation to return the loaned samples.

Once **DDSSubscriber::delete_contained_entities** (p. 1871) completes successfully, the application may delete the **DDSSubscriber** (p. 1860).

MT Safety:

UNSAFE. It is not safe to delete an entity while another thread may be simultaneously calling an API that uses the entity.

Exceptions

One	of the **Standard Return Codes** (p. 532), or **DDS_RETCODE_PRECONDITION_NOT_MET** (p. 532)

9.289.2.13 lookup_datareader()

```
virtual DDSDataReader * DDSSubscriber::lookup_datareader (
            const char * topic_name ) [pure virtual]
```

Retrieves an existing **DDSDataReader** (p. 1556).

Use this operation on the built-in **DDSSubscriber** (p. 1860) (**Built-in Topics** (p. 61)) to access the built-in **DDSData↩ Reader** (p. 1556) entities for the built-in topics.

The built-in **DDSDataReader** (p. 1556) is created when this operation is called on a built-in topic for the first time. The built-in **DDSDataReader** (p. 1556) is deleted automatically when the **DDSDomainParticipant** (p. 1619) is deleted.

To ensure that builtin **DDSDataReader** (p. 1556) entities receive all the discovery traffic, it is suggested that you lookup the builtin **DDSDataReader** (p. 1556) before the **DDSDomainParticipant** (p. 1619) is enabled. Looking up builtin **DDSDataReader** (p. 1556) may implicitly register builtin transports due to creation of **DDSDataReader** (p. 1556) (see **Built-in Transport Plugins** (p. 182) for details on when a builtin transport is registered). Therefore, if you are want to modify builtin transport properties, do so *before* using this operation.

Therefore the suggested sequence when looking up builtin DataReaders is:

- Create a disabled **DDSDomainParticipant** (p. 1619).
- (optional) Modify builtin transport properties
- Call **DDSDomainParticipant::get_builtin_subscriber()** (p. 1661).
- Call **DDSSubscriber::lookup_datareader()** (p. 1872).
- Call **enable()** (p. 1734) on the DomainParticipant.

Parameters

topic_name	<<***in***>> *(p. 433)* Name of the **DDSTopicDescription** (p. 1892) that the retrieved **DDSDataReader** (p. 1556) is attached to. Cannot be NULL.

Returns

A **DDSDataReader** (p. 1556) that belongs to the **DDSSubscriber** (p. 1860) attached to the **DDSTopicDescription** (p. 1892) with `topic_name`. If no such **DDSDataReader** (p. 1556) exists, this operation returns NULL.

The returned **DDSDataReader** (p. 1556) may be enabled or disabled.

If more than one **DDSDataReader** (p. 1556) is attached to the **DDSSubscriber** (p. 1860), this operation may return any one of them.

MT Safety:

UNSAFE. It is not safe to lookup a **DDSDataReader** (p. 1556) in one thread while another thread is simultaneously creating or destroying that **DDSDataReader** (p. 1556).

9.289.2.14 begin_access()

```
virtual DDS_ReturnCode_t DDSSubscriber::begin_access ( ) [pure virtual]
```

Indicates that the application is about to access the data samples in any of the **DDSDataReader** (p. 1556) objects attached to the **DDSSubscriber** (p. 1860).

If the **DDS_PresentationQosPolicy::access_scope** (p. 1252) of the **DDSSubscriber** (p. 1860) is **DDS_GROUP_PRESENTATION_QOS** (p. 616) or **DDS_HIGHEST_OFFERED_PRESENTATION_QOS** (p. 616) and **DDS_PresentationQosPolicy::ordered_access** (p. 1252) is **DDS_BOOLEAN_TRUE** (p. 513), the application is required to use this operation to access the samples in order across DataWriters of the same group (**DDSPublisher** (p. 1818) with **DDS_PresentationQosPolicy::access_scope** (p. 1252) set to **DDS_GROUP_PRESENTATION_QOS** (p. 616)).

In the above case, this operation must be called prior to calling any of the sample-accessing operations, **DDSSubscriber::get_datareaders** (p. 1874) on the **DDSSubscriber** (p. 1860), and **FooDataReader::read** (p. 1919), **FooDataReader::take** (p. 1920), **FooDataReader::read_w_condition** (p. 1925), and **FooDataReader::take_w_condition** (p. 1926) on any **DDSDataReader** (p. 1556).

Once the application has finished accessing the data samples, it must call **DDSSubscriber::end_access** (p. 1874)

The application is not required to call **DDSSubscriber::begin_access** (p. 1873) / **DDSSubscriber::end_access** (p. 1874) to access the samples in order if the **PRESENTATION** (p. 615) policy in the **DDSPublisher** (p. 1818) has **DDS_PresentationQosPolicy::access_scope** (p. 1252) set to something other than **DDS_GROUP_PRESENTATION_QOS** (p. 616). In this case, calling **DDSSubscriber::begin_access** (p. 1873) / **DDSSubscriber::end_access** (p. 1874) is not considered an error and has no effect.

Calls to **DDSSubscriber::begin_access** (p. 1873) / **DDSSubscriber::end_access** (p. 1874) may be nested and must be balanced.

Exceptions

One	of the **Standard Return Codes** (p. 532) or **DDS_RETCODE_NOT_ENABLED** (p. 533).

See also

Access to data samples (p. 129)

DDSSubscriber::get_datareaders (p. 1874)

PRESENTATION (p. 615)

9.289.2.15 end_access()

```
virtual DDS_ReturnCode_t DDSSubscriber::end_access ( ) [pure virtual]
```

Indicates that the application has finished accessing the data samples in **DDSDataReader** (p. 1556) objects managed by the **DDSSubscriber** (p. 1860).

This operation must be used to close a corresponding **begin_access()** (p. 1873).

This call must close a previous call to **DDSSubscriber::begin_access()** (p. 1873), otherwise the operation will fail with the error **DDS_RETCODE_PRECONDITION_NOT_MET** (p. 532).

Exceptions

One	of the **Standard Return Codes** (p. 532) or **DDS_RETCODE_PRECONDITION_NOT_MET** (p. 532) or **DDS_RETCODE_NOT_ENABLED** (p. 533).

9.289.2.16 get_datareaders()

```
virtual DDS_ReturnCode_t DDSSubscriber::get_datareaders (
              DDSDataReaderSeq & readers,
              DDS_SampleStateMask sample_states,
              DDS_ViewStateMask view_states,
              DDS_InstanceStateMask instance_states ) [pure virtual]
```

Allows the application to access the **DDSDataReader** (p. 1556) objects that contain samples with the specified `sample_states`, `view_states` and `instance_states`.

If the application is outside a **begin_access()** (p. 1873)/end_access() block, or if the **DDS_PresentationQosPolicy**↩ **::access_scope** (p. 1252) of the **DDSSubscriber** (p. 1860) is **DDS_INSTANCE_PRESENTATION_QOS** (p. 616) or **DDS_TOPIC_PRESENTATION_QOS** (p. 616), or if the **DDS_PresentationQosPolicy::ordered_access** (p. 1252) of

the **DDSSubscriber** (p. 1860) is **DDS_BOOLEAN_FALSE** (p. 513), the returned collection is a 'set' containing each **DDSDataReader** (p. 1556) at most once, in no specified order.

If the application is within a **begin_access()** (p. 1873)/end_access() block, and the **PRESENTATION** (p. 615) policy of the **DDSSubscriber** (p. 1860) is **DDS_GROUP_PRESENTATION_QOS** (p. 616) or **DDS_HIGHEST_OFFERED_PRESENTATION_QOS** (p. 616), and **DDS_PresentationQosPolicy::ordered_access** (p. 1252) in the **DDSSubscriber** (p. 1860) is **DDS_BOOLEAN_TRUE** (p. 513), the returned collection is a 'list' of DataReaders where a DataReader may appear more than one time.

To retrieve the samples in the order they were published across DataWriters of the same group (**DDSPublisher** (p. 1818) configured with **DDS_GROUP_PRESENTATION_QOS** (p. 616)), the application should read()/take() from each DataReader in the same order as it appears in the output sequence. The application will move to the next DataReader when the read()/take() operation fails with **DDS_RETCODE_NO_DATA** (p. 533).

Parameters

readers	<<***inout***>> *(p. 433)* a **DDSDataReaderSeq** (p. 1585) object where the set or list of readers will be returned.
sample_states	<<***in***>> *(p. 433)* the returned DataReader must contain samples that have one of these `sample_states`.
view_states	<<***in***>> *(p. 433)* the returned DataReader must contain samples that have one of these `view_states`.
instance_states	<<***in***>> *(p. 433)* the returned DataReader must contain samples that have one of these `instance_states`.

Exceptions

One	of the **Standard Return Codes** (p. 532) or **DDS_RETCODE_NOT_ENABLED** (p. 533).

See also

Access to data samples (p. 129)

DDSSubscriber::begin_access (p. 1873)

DDSSubscriber::end_access (p. 1874)

PRESENTATION (p. 615)

9.289.2.17 get_all_datareaders()

```
virtual DDS_ReturnCode_t DDSSubscriber::get_all_datareaders (
            DDSDataReaderSeq & readers ) [pure virtual]
```

Retrieve all the DataReaders created from this Subscriber.

Parameters

readers	<<***inout***>> *(p. 433)* Sequence where the DataReaders will be added

Exceptions

One	of the **Standard Return Codes** (p. 532)

9.289.2.18 notify_datareaders()

```
virtual DDS_ReturnCode_t DDSSubscriber::notify_datareaders ( ) [pure virtual]
```

Invokes the operation **DDSDataReaderListener::on_data_available()** (p. 1585) on the **DDSDataReaderListener** (p. 1583) objects attached to contained **DDSDataReader** (p. 1556) entities with **DDS_DATA_AVAILABLE_STATUS** (p. 541) that is considered changed as described in **Changes in read communication status** (p. 536).

This operation is typically invoked from the **DDSSubscriberListener::on_data_on_readers** (p. 1882) operation in the **DDSSubscriberListener** (p. 1881). That way the **DDSSubscriberListener** (p. 1881) can delegate to the **DDSData↩ ReaderListener** (p. 1583) objects the handling of the data.

The operation will notify the data readers that have a `sample_state` of **DDS_NOT_READ_SAMPLE_STATE** (p. 162), `view_state` of **DDS_ANY_SAMPLE_STATE** (p. 162) and `instance_state` of **DDS_ANY_INSTANCE_STATE** (p. 166).

Exceptions

One	of the **Standard Return Codes** (p. 532), or **DDS_RETCODE_NOT_ENABLED** (p. 533).

9.289.2.19 get_participant()

```
virtual DDSDomainParticipant * DDSSubscriber::get_participant ( ) [pure virtual]
```

Returns the **DDSDomainParticipant** (p. 1619) to which the **DDSSubscriber** (p. 1860) belongs.

Returns

the **DDSDomainParticipant** (p. 1619) to which the **DDSSubscriber** (p. 1860) belongs.

9.289.2.20 copy_from_topic_qos()

```
virtual  DDS_ReturnCode_t DDSSubscriber::copy_from_topic_qos (
              DDS_DataReaderQos & datareader_qos,
            const  DDS_TopicQos & topic_qos )  [pure virtual]
```

Copies the policies in the **DDS_TopicQos** (p. 1397) to the corresponding policies in the **DDS_DataReaderQos** (p. 858).

Copies the policies in the **DDS_TopicQos** (p. 1397) to the corresponding policies in the **DDS_DataReaderQos** (p. 858) (replacing values in the **DDS_DataReaderQos** (p. 858), if present).

This is a "convenience" operation most useful in combination with the operations **DDSSubscriber::get_default_↩ datareader_qos** (p. 1863) and **DDSTopic::get_qos** (p. 1889). The operation **DDSSubscriber::copy_from_topic_qos** (p. 1876) can be used to merge the **DDSDataReader** (p. 1556) default QoS policies with the corresponding ones on the **DDSTopic** (p. 1885). The resulting QoS can then be used to create a new **DDSDataReader** (p. 1556), or set its QoS.

This operation does not check the resulting **DDS_DataReaderQos** (p. 858) for consistency. This is because the 'merged' **DDS_DataReaderQos** (p. 858) may not be the final one, as the application can still modify some policies prior to applying the policies to the **DDSDataReader** (p. 1556).

Parameters

datareader_qos	*<<**inout**>> (p. 433)* **DDS_DataReaderQos** (p. 858) to be filled-up.
topic_qos	*<<**in**>> (p. 433)* **DDS_TopicQos** (p. 1397) to be merged with **DDS_DataReaderQos** (p. 858).

Exceptions

One	of the **Standard Return Codes** (p. 532)

9.289.2.21 set_qos()

```
virtual  DDS_ReturnCode_t DDSSubscriber::set_qos (
            const  DDS_SubscriberQos & qos )  [pure virtual]
```

Sets the subscriber QoS.

This operation modifies the QoS of the **DDSSubscriber** (p. 1860).

The **DDS_SubscriberQos::group_data** (p. 1369), **DDS_SubscriberQos::partition** (p. 1369) and **DDS_Subscriber↩ Qos::entity_factory** (p. 1369) can be changed. The other policies are immutable.

Parameters

qos	<<***in***>> *(p. 433)* **DDS_SubscriberQos** (p. 1367) to be set to. Policies must be consistent. Immutable policies cannot be changed after **DDSSubscriber** (p. 1860) is enabled. The special value **DDS_SUBSCRIBER_QOS_DEFAULT** (p. 59) can be used to indicate that the QoS of the **DDSSubscriber** (p. 1860) should be changed to match the current default **DDS_SubscriberQos** (p. 1367) set in the **DDSDomainParticipant** (p. 1619).

Exceptions

One	of the **Standard Return Codes** (p. 532), **DDS_RETCODE_IMMUTABLE_POLICY** (p. 533), or **DDS_RETCODE_INCONSISTENT_POLICY** (p. 533).

See also

DDS_SubscriberQos (p. 1367) for rules on consistency among QoS

set_qos (abstract) (p. 1732)

Operations Allowed in Listener Callbacks (p. 1796)

9.289.2.22 set_qos_with_profile()

```
virtual  DDS_ReturnCode_t DDSSubscriber::set_qos_with_profile (
             const char * library_name,
             const char * profile_name )  [pure virtual]
```

<<***extension***>> *(p. 432)* Change the QoS of this subscriber using the input XML QoS profile.

This operation modifies the QoS of the **DDSSubscriber** (p. 1860).

The **DDS_SubscriberQos::group_data** (p. 1369), **DDS_SubscriberQos::partition** (p. 1369) and **DDS_Subscriber↩ Qos::entity_factory** (p. 1369) can be changed. The other policies are immutable.

Parameters

library_name	<<***in***>> *(p. 433)* Library name containing the XML QoS profile. If library_name is null RTI Connext will use the default library (see **DDSSubscriber::set_default_library** (p. 1865)).
profile_name	<<***in***>> *(p. 433)* XML QoS Profile name. If profile_name is null RTI Connext will use the default profile (see **DDSSubscriber::set_default_profile** (p. 1866)).

Exceptions

One	of the **Standard Return Codes** (p. 532), **DDS_RETCODE_IMMUTABLE_POLICY** (p. 533), or **DDS_RETCODE_INCONSISTENT_POLICY** (p. 533).

See also

DDS_SubscriberQos (p. 1367) for rules on consistency among QoS

Operations Allowed in Listener Callbacks (p. 1796)

9.289.2.23 get_qos()

```
virtual DDS_ReturnCode_t DDSSubscriber::get_qos (
            DDS_SubscriberQos & qos ) [pure virtual]
```

Gets the subscriber QoS.

This method may potentially allocate memory depending on the sequences contained in some QoS policies.

Parameters

qos	<<***in***>> *(p. 433)* **DDS_SubscriberQos** (p. 1367) to be filled in.

Exceptions

One	of the **Standard Return Codes** (p. 532)

See also

get_qos (abstract) (p. 1733)

9.289.2.24 set_listener()

```
virtual DDS_ReturnCode_t DDSSubscriber::set_listener (
            DDSSubscriberListener * l,
            DDS_StatusMask mask = DDS_STATUS_MASK_ALL ) [pure virtual]
```

Sets the subscriber listener.

Parameters

l	<<***in***>> *(p. 433)* **DDSSubscriberListener** (p. 1881) to set to.
mask	<<***in***>> *(p. 433)* **DDS_StatusMask** (p. 537) associated with the **DDSSubscriberListener** (p. 1881).

Exceptions

One	of the **Standard Return Codes** (p. 532)

See also

set_listener (abstract) (p. 1733)

9.289.2.25 get_listener()

```
virtual DDSSubscriberListener * DDSSubscriber::get_listener ( ) [pure virtual]
```

Get the subscriber listener.

Returns

DDSSubscriberListener (p. 1881) of the **DDSSubscriber** (p. 1860).

See also

get_listener (abstract) (p. 1734)

9.289.2.26 lookup_datareader_by_name()

```
virtual DDSDataReader * DDSSubscriber::lookup_datareader_by_name (
            const char * datareader_name ) [pure virtual]
```

<<***extension***>> *(p. 432)* Retrieves a **DDSDataReader** (p. 1556) contained within the **DDSSubscriber** (p. 1860) the **DDSDataReader** (p. 1556) entity name.

Every **DDSDataReader** (p. 1556) in the system has an entity name which is configured and stored in the <<***extension***>> *(p. 432)* EntityName policy, **ENTITY_NAME** (p. 601).

This operation retrieves the **DDSDataReader** (p. 1556) within the **DDSSubscriber** (p. 1860) whose name matches the one specified. If there are several **DDSDataReader** (p. 1556) with the same name within the **DDSSubscriber** (p. 1860), the operation returns the first matching occurrence.

Parameters

datareader_name	<<***in***>> *(p. 433)* Entity name of the **DDSDataReader** (p. 1556).

Returns

The first **DDSDataReader** (p. 1556) found with the specified name or NULL if it is not found.

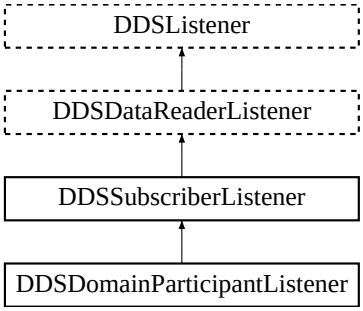
See also

DDSDomainParticipant::lookup_datareader_by_name (p. 1692)

9.290 DDSSubscriberListener Class Reference

<<***interface***>> *(p. 433)* **DDSListener** (p. 1794) for status about a subscriber.

Inheritance diagram for DDSSubscriberListener:



Public Member Functions

- virtual void **on_data_on_readers** (**DDSSubscriber** ∗sub)

  *Handles the **DDS_DATA_ON_READERS_STATUS** (p. 541) communication status.*

9.290.1 Detailed Description

<<***interface***>> *(p. 433)* **DDSListener** (p. 1794) for status about a subscriber.

Entity:

DDSSubscriber (p. 1860)

Status:

DDS_DATA_AVAILABLE_STATUS (p. 541);
DDS_DATA_ON_READERS_STATUS (p. 541);
DDS_LIVELINESS_CHANGED_STATUS (p. 542), **DDS_LivelinessChangedStatus** (p. 1176);
DDS_REQUESTED_DEADLINE_MISSED_STATUS (p. 540), **DDS_RequestedDeadlineMissedStatus** (p. 1306);
DDS_REQUESTED_INCOMPATIBLE_QOS_STATUS (p. 540), **DDS_RequestedIncompatibleQosStatus** (p. 1307);
DDS_SAMPLE_LOST_STATUS (p. 541), **DDS_SampleLostStatus** (p. 1355);
DDS_SAMPLE_REJECTED_STATUS (p. 541), **DDS_SampleRejectedStatus** (p. 1356);
DDS_SUBSCRIPTION_MATCHED_STATUS (p. 543), **DDS_SubscriptionMatchedStatus** (p. 1379);

See also

DDSListener (p. 1794)

Status Kinds (p. 533)

Operations Allowed in Listener Callbacks (p. 1796)

9.290.2 Member Function Documentation

9.290.2.1 on_data_on_readers()

```
virtual void DDSSubscriberListener::on_data_on_readers (
              DDSSubscriber * sub )  [virtual]
```

Handles the **DDS_DATA_ON_READERS_STATUS** (p. 541) communication status.

9.291 DDSSubscriberSeq Class Reference

Declares IDL `sequence` < **DDSSubscriber** (p. 1860) > .

9.291.1 Detailed Description

Declares IDL `sequence` < **DDSSubscriber** (p. 1860) > .

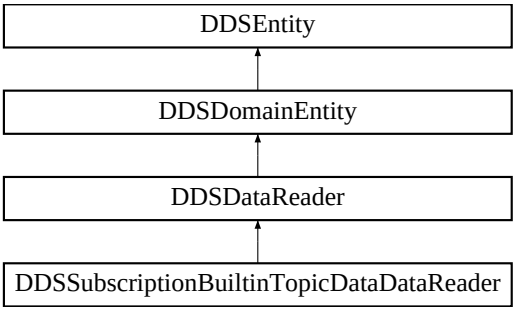
See also

FooSeq (p. 1965)

9.292 DDSSubscriptionBuiltinTopicDataDataReader Class Reference

Instantiates `DataReader` < **DDS_SubscriptionBuiltinTopicData** (p. 1370) > .

Inheritance diagram for DDSSubscriptionBuiltinTopicDataDataReader:



Additional Inherited Members

9.292.1 Detailed Description

Instantiates `DataReader` < **DDS_SubscriptionBuiltinTopicData** (p. 1370) > .

DDSDataReader (p. 1556) of topic **DDS_SUBSCRIPTION_TOPIC_NAME** (p. 496) used for accessing **DDS_↩ SubscriptionBuiltinTopicData** (p. 1370) of the remote **DDSDataReader** (p. 1556) and the associated **DDSSubscriber** (p. 1860).

Instantiates:

<<***generic***>> *(*p. *433)* **FooDataReader** (p. 1917)

See also

DDS_SubscriptionBuiltinTopicData (p. 1370)

DDS_SUBSCRIPTION_TOPIC_NAME (p. 496)

9.293 DDSSubscriptionBuiltinTopicDataTypeSupport Class Reference

Instantiates `TypeSupport` < **DDS_SubscriptionBuiltinTopicData** (p. 1370) > .

9.293.1 Detailed Description

Instantiates `TypeSupport` < **DDS_SubscriptionBuiltinTopicData** (p. 1370) > .

Instantiates:

<<***generic***>> *(*p. *433)* **FooTypeSupport** (p. 1979)

See also

DDS_SubscriptionBuiltinTopicData (p. 1370)

9.294 DDSThreadFactory Class Reference

<<***extension***>> *(*p. *432)* <<***interface***>> *(*p. *433)* Interface for providing the threads needed by the middleware. It consists of operations to create and delete threads. The interface depends on the **DDSThreadFactory_OnSpawned↩ Function** (p. 718) that specifies the operation to run in the new thread.

Inherited by NDDSStackManagedThreadFactory.

Public Member Functions

- virtual void * **create_thread** (const char *thread_name, const struct **DDS_ThreadSettings_t** &settings, **DDSThreadFactory_OnSpawnedFunction** on_spawned, void *thread_param)=0

  Handles the creation of new threads.
- virtual void **delete_thread** (void *thread)=0

  Handles the deletion of threads previously created by this factory.

9.294.1 Detailed Description

<<***extension***>> *(*p. *432)* <<***interface***>> *(*p. *433)* Interface for providing the threads needed by the middleware. It consists of operations to create and delete threads. The interface depends on the **DDSThreadFactory_OnSpawned↩ Function** (p. 718) that specifies the operation to run in the new thread.

9.294.2 Member Function Documentation

9.294.2.1 create_thread()

```
virtual void * DDSThreadFactory::create_thread (
            const char * thread_name,
            const struct  DDS_ThreadSettings_t & settings,
             DDSThreadFactory_OnSpawnedFunction on_spawned,
            void * thread_param )  [pure virtual]
```

Handles the creation of new threads.

This callback is called by the middleware whenever it needs to create a new thread. The operation creates a new thread of control that will call the function specified by **DDSThreadFactory_OnSpawnedFunction** (p. 718). On success, this operation must guarantee that after return the new thread of control been spawned and its execution has started or will start some time after.

The **DDSThreadFactory_OnSpawnedFunction** (p. 718), function must be called in the new thread and return its value on finalization. The function should be called as follows:

```
void * thread_out = on_spawned(thread_params);

...

return thread_out;
```

Some thread frameworks do this directly by just receiving the function and its parameters as arguments on thread creation. For instance, POSIX follows this model when creating a thread and the start_routine function has the same signature as **DDSThreadFactory_OnSpawnedFunction** (p. 718), so the parameter can be passed directly:

```
int result = pthread_create(&pthread_handle, &attr, on_spawned, thread_params);

...

return &pthread_handle;
```

9.294.2.2 delete_thread()

```
virtual void DDSThreadFactory::delete_thread (
            void * thread )  [pure virtual]
```

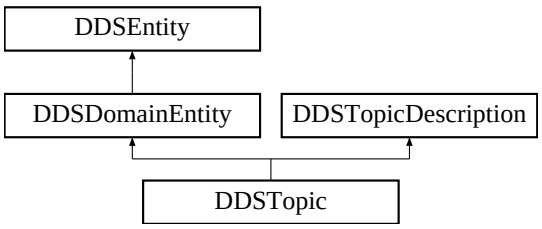
Handles the deletion of threads previously created by this factory.

This callback is called by the middleware whenever it needs to delete a thread. This operation deletes a thread previously created via a call to **DDSThreadFactory::create_thread** (p. 1884) on the same factory object. On success, the thread must be deleted but it does not have to guarantee that the execution has finished. Likewise, resources associated with the underlying operating system may be still in use. Depending on the thread framework, resources allocated on thread creation are released upon thread finalization. POSIX follows this model so no actions are required on the thread handle returned by pthread_create().

9.295 DDSTopic Class Reference

<<***interface***>> *(p. 433)* The most basic description of the data to be published and subscribed.

Inheritance diagram for DDSTopic:



Public Member Functions

- virtual **DDS_ReturnCode_t** **get_inconsistent_topic_status** (**DDS_InconsistentTopicStatus** &status)=0

  *Allows the application to retrieve the **DDS_INCONSISTENT_TOPIC_STATUS** (p. 539) status of a **DDSTopic** (p. 1885).*
- virtual **DDS_ReturnCode_t** **set_qos** (const **DDS_TopicQos** &qos)=0

  Set the topic QoS.
- virtual **DDS_ReturnCode_t** **set_qos_with_profile** (const char *library_name, const char *profile_name)=0

  <<***extension***>> *(p. 432) Change the QoS of this topic using the input XML QoS profile.*
- virtual **DDS_ReturnCode_t** **get_qos** (**DDS_TopicQos** &qos)=0

  Get the topic QoS.
- virtual **DDS_ReturnCode_t** **set_listener** (**DDSTopicListener** *l, **DDS_StatusMask** mask= **DDS_STATUS_MASK_ALL**)=0

  Set the topic listener.
- virtual **DDSTopicListener** * **get_listener** ()=0

  Get the topic listener.

Static Public Member Functions

- static **DDSTopic** * **narrow** (**DDSTopicDescription** *topic_description)

  *Narrow the given **DDSTopicDescription** (p. 1892) pointer to a **DDSTopic** (p. 1885) pointer.*

9.295.1 Detailed Description

<<***interface***>> *(p. 433)* The most basic description of the data to be published and subscribed.

QoS:

DDS_TopicQos (p. 1397)

Status:

DDS_INCONSISTENT_TOPIC_STATUS (p. 539), **DDS_InconsistentTopicStatus** (p. 1165)

Listener:

DDSTopicListener (p. 1894)

A **DDSTopic** (p. 1885) is identified by its name, which must be unique in the whole domain. In addition (by virtue of extending **DDSTopicDescription** (p. 1892)) it fully specifies the type of the data that can be communicated when publishing or subscribing to the **DDSTopic** (p. 1885).

DDSTopic (p. 1885) is the only **DDSTopicDescription** (p. 1892) that can be used for publications and therefore associated with a **DDSDataWriter** (p. 1589).

The following operations may be called even if the **DDSTopic** (p. 1885) is not enabled. Other operations will fail with the value **DDS_RETCODE_NOT_ENABLED** (p. 533) if called on a disabled **DDSTopic** (p. 1885):

- **DDSEntity::enable** (p. 1734)
- **DDSTopic::set_qos** (p. 1887), **DDSTopic::get_qos** (p. 1889)
- **DDSTopic::set_qos_with_profile** (p. 1888)
- **DDSTopic::set_listener** (p. 1889), **DDSTopic::get_listener** (p. 1890)
- **DDSEntity::get_statuscondition** (p. 1735), **DDSEntity::get_status_changes** (p. 1735)
- **DDSTopic::get_inconsistent_topic_status** (p. 1887)

See also

Operations Allowed in Listener Callbacks (p. 1796)

Examples

HelloWorld_publisher.cxx, and **HelloWorld_subscriber.cxx**.

9.295.2 Member Function Documentation

9.295.2.1 narrow()

```
static  DDSTopic * DDSTopic::narrow (
            DDSTopicDescription * topic_description )  [static]
```

Narrow the given **DDSTopicDescription** (p. 1892) pointer to a **DDSTopic** (p. 1885) pointer.

Returns

DDSTopic (p. 1885) if this **DDSTopicDescription** (p. 1892) is a **DDSTopic** (p. 1885). Otherwise, return NULL.

9.295.2.2 get_inconsistent_topic_status()

```
virtual  DDS_ReturnCode_t DDSTopic::get_inconsistent_topic_status (
            DDS_InconsistentTopicStatus & status )  [pure virtual]
```

Allows the application to retrieve the **DDS_INCONSISTENT_TOPIC_STATUS** (p. 539) status of a **DDSTopic** (p. 1885).

Retrieve the current **DDS_InconsistentTopicStatus** (p. 1165)

Parameters

status	<<***inout***>> *(p. 433)* Status to be retrieved.

Exceptions

One	of the **Standard Return Codes** (p. 532)

See also

DDS_InconsistentTopicStatus (p. 1165)

9.295.2.3 set_qos()

```
virtual  DDS_ReturnCode_t DDSTopic::set_qos (
            const  DDS_TopicQos & qos )  [pure virtual]
```

Set the topic QoS.

The **DDS_TopicQos::topic_data** (p. 1400) and **DDS_TopicQos::deadline** (p. 1400), **DDS_TopicQos::latency_budget** (p. 1401), **DDS_TopicQos::transport_priority** (p. 1402) and **DDS_TopicQos::lifespan** (p. 1402) can be changed. The other policies are immutable.

Parameters

qos	<<***in***>> *(p. 433)* Set of policies to be applied to **DDSTopic** (p. 1885).

Policies must be consistent. Immutable policies cannot be changed after **DDSTopic** (p. 1885) is enabled. The special value **DDS_TOPIC_QOS_DEFAULT** (p. 58) can be used to indicate that the QoS of the **DDSTopic** (p. 1885) should be changed to match the current default **DDS_TopicQos** (p. 1397) set in the **DDSDomainParticipant** (p. 1619).

Exceptions

One	of the **Standard Return Codes** (p. 532), **DDS_RETCODE_IMMUTABLE_POLICY** (p. 533) if immutable policy is changed, or **DDS_RETCODE_INCONSISTENT_POLICY** (p. 533) if policies are inconsistent

See also

DDS_TopicQos (p. 1397) for rules on consistency among QoS

set_qos (abstract) (p. 1732)

Operations Allowed in Listener Callbacks (p. 1796)

9.295.2.4 set_qos_with_profile()

```
virtual  DDS_ReturnCode_t DDSTopic::set_qos_with_profile (
             const char * library_name,
             const char * profile_name )  [pure virtual]
```

<<***extension***>> *(p. 432)* Change the QoS of this topic using the input XML QoS profile.

The **DDS_TopicQos::topic_data** (p. 1400) and **DDS_TopicQos::deadline** (p. 1400), **DDS_TopicQos::latency_budget** (p. 1401), **DDS_TopicQos::transport_priority** (p. 1402) and **DDS_TopicQos::lifespan** (p. 1402) can be changed. The other policies are immutable.

Parameters

library_name	<<***in***>> *(p. 433)* Library name containing the XML QoS profile. If library_name is null RTI Connext will use the default library (see **DDSDomainParticipant::set_default_library** (p. 1635)).
profile_name	<<***in***>> *(p. 433)* XML QoS Profile name. If profile_name is null RTI Connext will use the default profile (see **DDSDomainParticipant::set_default_profile** (p. 1636)).

Exceptions

One	of the **Standard Return Codes** (p. 532), **DDS_RETCODE_IMMUTABLE_POLICY** (p. 533) if immutable policy is changed, or **DDS_RETCODE_INCONSISTENT_POLICY** (p. 533) if policies are inconsistent

See also

DDS_TopicQos (p. 1397) for rules on consistency among QoS

Operations Allowed in Listener Callbacks (p. 1796)

9.295.2.5 get_qos()

```
virtual  DDS_ReturnCode_t DDSTopic::get_qos (
              DDS_TopicQos & qos )  [pure virtual]
```

Get the topic QoS.

This method may potentially allocate memory depending on the sequences contained in some QoS policies.

Parameters

qos	<<***inout***>> *(p. 433)* QoS to be filled up.

Exceptions

One	of the **Standard Return Codes** (p. 532)

See also

get_qos (abstract) (p. 1733)

9.295.2.6 set_listener()

```
virtual  DDS_ReturnCode_t DDSTopic::set_listener (
              DDSTopicListener * l,
              DDS_StatusMask mask =  DDS_STATUS_MASK_ALL )  [pure virtual]
```

Set the topic listener.

Parameters

l	<<***in***>> *(p. 433)* Listener to be installed on entity.
mask	<<***in***>> *(p. 433)* Changes of communication status to be invoked on the listener. See **DDS_StatusMask** (p. 537).

Exceptions

One	of the **Standard Return Codes** (p. 532)

See also

set_listener (abstract) (p. 1733)

9.295.2.7 get_listener()

```
virtual DDSTopicListener * DDSTopic::get_listener ( ) [pure virtual]
```

Get the topic listener.

Returns

Existing listener attached to the **DDSTopic** (p. 1885).

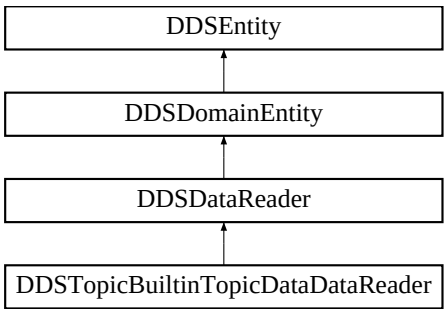
See also

get_listener (abstract) (p. 1734)

9.296 DDSTopicBuiltinTopicDataDataReader Class Reference

Instantiates `DataReader` < **DDS_TopicBuiltinTopicData** (p. 1391) > .

Inheritance diagram for DDSTopicBuiltinTopicDataDataReader:



Additional Inherited Members

9.296.1 Detailed Description

Instantiates `DataReader` < **DDS_TopicBuiltinTopicData** (p. 1391) > .

DDSDataReader (p. 1556) of topic **DDS_TOPIC_TOPIC_NAME** (p. 493) used for accessing **DDS_TopicBuiltinTopic↩Data** (p. 1391) of the remote **DDSTopic** (p. 1885).

Note: The **DDS_TopicBuiltinTopicData** (p. 1391) built-in topic is meant to convey information about discovered Topics. This Topic's samples are not propagated in a separate packet on the wire. Instead, the data is sent as part of the information carried by other built-in topics (**DDS_PublicationBuiltinTopicData** (p. 1264) and **DDS_Subscription↩BuiltinTopicData** (p. 1370)). Therefore TopicBuiltinTopicData DataReaders will not receive any data.

Instantiates:

<<***generic***>> *(p. 433)* **FooDataReader** (p. 1917)

See also

DDS_TopicBuiltinTopicData (p. 1391)

DDS_TOPIC_TOPIC_NAME (p. 493)

9.297 DDSTopicBuiltinTopicDataTypeSupport Class Reference

Instantiates `TypeSupport` < **DDS_TopicBuiltinTopicData** (p. 1391) > .

9.297.1 Detailed Description

Instantiates `TypeSupport` < **DDS_TopicBuiltinTopicData** (p. 1391) > .

Instantiates:

<<***generic***>> *(p. 433)* **FooTypeSupport** (p. 1979)

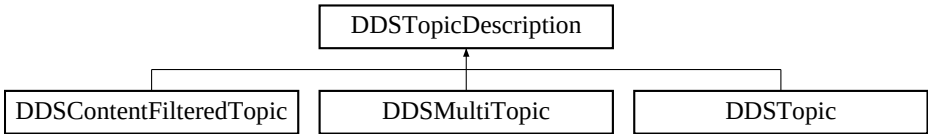
See also

DDS_TopicBuiltinTopicData (p. 1391)

9.298 DDSTopicDescription Class Reference

<<***interface***>> *(*p. *433)* Base class for **DDSTopic** (p. 1885), **DDSContentFilteredTopic** (p. 1551), and **DDSMulti↩ Topic** (p. 1797).

Inheritance diagram for DDSTopicDescription:



Public Member Functions

- virtual const char * **get_type_name** ()=0

  Get the associated `type_name`*.*
- virtual const char * **get_name** ()=0

  Get the name used to create this ***DDSTopicDescription*** *(*p. *1892) .*
- virtual **DDSDomainParticipant** * **get_participant** ()=0

  Get the ***DDSDomainParticipant*** *(*p. *1619) to which the* ***DDSTopicDescription*** *(*p. *1892) belongs.*

9.298.1 Detailed Description

<<***interface***>> *(*p. *433)* Base class for **DDSTopic** (p. 1885), **DDSContentFilteredTopic** (p. 1551), and **DDSMulti↩ Topic** (p. 1797).

DDSTopicDescription (p. 1892) represents the fact that both publications and subscriptions are tied to a single data-type. Its attribute `type_name` defines a unique resulting type for the publication or the subscription and therefore creates an implicit association with a **DDSTypeSupport** (p. 1897).

DDSTopicDescription (p. 1892) has also a `name` that allows it to be retrieved locally.

See also

DDSTypeSupport (p. 1897), **FooTypeSupport** (p. 1979)

9.298.2 Member Function Documentation

9.298.2.1 get_type_name()

```
virtual const char * DDSTopicDescription::get_type_name ( )  [pure virtual]
```

Get the associated `type_name`.

The type name defines a locally unique type for the publication or the subscription.

The `type_name` corresponds to a unique string used to register a type via the **FooTypeSupport::register_type** (p. 1981) method.

Thus, the `type_name` implies an association with a corresponding **DDSTypeSupport** (p. 1897) and this **DDSTopic↩ Description** (p. 1892).

Returns

the type name. The returned type name is valid until the **DDSTopicDescription** (p. 1892) is deleted.

Postcondition

The result is non-NULL.

See also

DDSTypeSupport (p. 1897), **FooTypeSupport** (p. 1979)

9.298.2.2 get_name()

```
virtual const char * DDSTopicDescription::get_name ( )  [pure virtual]
```

Get the name used to create this **DDSTopicDescription** (p. 1892) .

Returns

the name used to create this **DDSTopicDescription** (p. 1892). The returned topic name is valid until the **DDSTopicDescription** (p. 1892) is deleted.

Postcondition

The result is non-NULL.

9.298.2.3 get_participant()

```
virtual DDSDomainParticipant * DDSTopicDescription::get_participant ( ) [pure virtual]
```

Get the **DDSDomainParticipant** (p. 1619) to which the **DDSTopicDescription** (p. 1892) belongs.

Returns

The **DDSDomainParticipant** (p. 1619) to which the **DDSTopicDescription** (p. 1892) belongs.

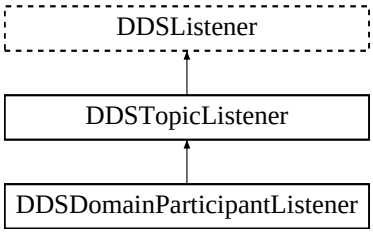
Postcondition

The result is non-NULL.

9.299 DDSTopicListener Class Reference

<<***interface***>> *(p. 433)* **DDSListener** (p. 1794) for **DDSTopic** (p. 1885) entities.

Inheritance diagram for DDSTopicListener:



Public Member Functions

- virtual void **on_inconsistent_topic** (**DDSTopic** *topic, const **DDS_InconsistentTopicStatus** &status)
  *Handle the **DDS_INCONSISTENT_TOPIC_STATUS** (p. 539) status.*

9.299.1 Detailed Description

<<***interface***>> *(p. 433)* **DDSListener** (p. 1794) for **DDSTopic** (p. 1885) entities.

Entity:

DDSTopic (p. 1885)

Status:

DDS_INCONSISTENT_TOPIC_STATUS (p. 539), **DDS_InconsistentTopicStatus** (p. 1165)

This is the interface that can be implemented by an application-provided class and then registered with the **DDSTopic** (p. 1885) such that the application can be notified by RTI Connext of relevant status changes.

See also

Status Kinds (p. 533)

DDSListener (p. 1794)

DDSTopic::set_listener (p. 1889)

Operations Allowed in Listener Callbacks (p. 1796)

9.299.2 Member Function Documentation

9.299.2.1 on_inconsistent_topic()

```
virtual void DDSTopicListener::on_inconsistent_topic (
             DDSTopic * topic,
            const  DDS_InconsistentTopicStatus & status )  [virtual]
```

Handle the **DDS_INCONSISTENT_TOPIC_STATUS** (p. 539) status.

This callback is called when a remote **DDSTopic** (p. 1885) is discovered but is inconsistent with the locally created **DDSTopic** (p. 1885) of the same topic name.

Parameters

topic	<<***out***>> *(p. 433)* Locally created **DDSTopic** (p. 1885) that triggers the listener callback
status	<<***out***>> *(p. 433)* Current inconsistent status of locally created **DDSTopic** (p. 1885)

9.300 DDSTopicQuery Class Reference

<<***extension***>> *(p. 432)* Allows a **DDSDataReader** (p. 1556) to query the sample cache of its matching **DDSDataWriter** (p. 1589).

Public Member Functions

- virtual **DDS_ReturnCode_t get_guid** (**DDS_GUID_t** &guid)
  Get the TopicQuery's GUID.

9.300.1 Detailed Description

<<***extension***>> *(p. 432)* Allows a **DDSDataReader** (p. 1556) to query the sample cache of its matching **DDSDataWriter** (p. 1589).

9.300.2 Member Function Documentation

9.300.2.1 get_guid()

```
virtual  DDS_ReturnCode_t DDSTopicQuery::get_guid (
              DDS_GUID_t & guid )  [inline], [virtual]
```

Get the TopicQuery's GUID.

Parameters

guid	<<***out***>> *(*p. *433)* The GUID of the **DDSTopicQuery** (p. 1895).

Exceptions

One	of the **Standard Return Codes** (p. 532)

9.301 DDSTopicQueryHelper Class Reference

Helpers to provide utility operations related to **DDSTopicQuery** (p. 1895).

Static Public Member Functions

- static **DDS_Boolean** **topic_query_data_from_service_request** (**DDS_TopicQueryData** &topic_query_data, const **DDS_ServiceRequest** &service_request)

  *Retrieves the **DDS_TopicQueryData** (*p. *1402) data contained in the specified **DDS_ServiceRequest** (*p. *1359).*

9.301.1 Detailed Description

Helpers to provide utility operations related to **DDSTopicQuery** (p. 1895).

9.301.2 Member Function Documentation

9.301.2.1 topic_query_data_from_service_request()

```
static  DDS_Boolean DDSTopicQueryHelper::topic_query_data_from_service_request (
              DDS_TopicQueryData & topic_query_data,
            const  DDS_ServiceRequest & service_request )  [static]
```

Retrieves the **DDS_TopicQueryData** (p. 1402) data contained in the specified **DDS_ServiceRequest** (p. 1359).

This operation will extract the content from the request body of the **DDS_ServiceRequest** (p. 1359) and place it into the specified **DDS_TopicQueryData** (p. 1402) object.

The specified **DDS_ServiceRequest** (p. 1359) must be a valid sample associated with the service id **DDS_TOPIC_↩ QUERY_SERVICE_REQUEST_ID** (p. 498). Otherwise this operation will return **DDS_BOOLEAN_FALSE** (p. 513).

This operation can be called within the context of a **DDSDataWriterListener::on_service_request_accepted** (p. 1618) to retrieve the **DDS_TopicQueryData** (p. 1402) of a **DDS_ServiceRequest** (p. 1359) that has been received with service id **DDS_TOPIC_QUERY_SERVICE_REQUEST_ID** (p. 498)

Parameters

topic_query_data	*<<**inout**>> (p. 433)* A **DDS_TopicQueryData** (p. 1402) object where the content from the service request is extracted.
service_request	*<<**in**>> (p. 433)* Input **DDS_ServiceRequest** (p. 1359) that contains the **DDS_TopicQueryData** (p. 1402) as part of its request body.

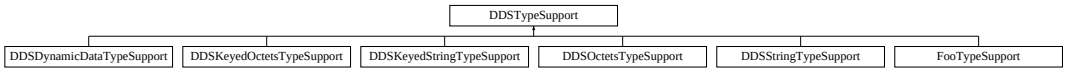
Returns

DDS_BOOLEAN_TRUE (p. 513) on success.

9.302 DDSTypeSupport Class Reference

*<<**interface**>> (p. 433)* An abstract *marker* interface that has to be specialized for each concrete user data type that will be used by the application.

Inheritance diagram for DDSTypeSupport:



9.302.1 Detailed Description

*<<**interface**>> (p. 433)* An abstract *marker* interface that has to be specialized for each concrete user data type that will be used by the application.

The implementation provides an automatic means to generate a type-specific class, **FooTypeSupport** (p. 1979), from a description of the type in IDL.

A **DDSTypeSupport** (p. 1897) must be registered using the **FooTypeSupport::register_type** (p. 1981) operation on this type-specific class before it can be used to create **DDSTopic** (p. 1885) objects.

See also

FooTypeSupport (p. 1979)

the `Code Generator User's Manual`

Examples

HelloWorldSupport.cxx.

9.303 DDSWaitSet Class Reference

*<<**interface**>> (p. 433)* Allows an application to wait until one or more of the attached **DDSCondition** (p. 1544) objects has a `trigger_value` of **DDS_BOOLEAN_TRUE** (p. 513) or else until the timeout expires.

Public Member Functions

- virtual **DDS_ReturnCode_t wait** (**DDSConditionSeq** &active_conditions, const **DDS_Duration_t** &timeout)

  Allows an application thread to wait for the occurrence of certain conditions.
- virtual **DDS_ReturnCode_t attach_condition** (**DDSCondition** *cond)

  *Attaches a **DDSCondition** (p. 1544) to the **DDSWaitSet** (p. 1897).*
- virtual **DDS_ReturnCode_t detach_condition** (**DDSCondition** *cond)

  *Detaches a **DDSCondition** (p. 1544) from the **DDSWaitSet** (p. 1897).*
- virtual **DDS_ReturnCode_t get_conditions** (**DDSConditionSeq** &attached_conditions)

  *Retrieves the list of attached **DDSCondition** (p. 1544) (s).*
- virtual **DDS_ReturnCode_t set_property** (const **DDS_WaitSetProperty_t** &prop)

  *<<**extension**>> (p. 432) Sets the **DDS_WaitSetProperty_t** (p. 1509), to configure the associated **DDSWaitSet** (p. 1897) to return after one or more trigger events have occurred.*
- virtual **DDS_ReturnCode_t get_property** (**DDS_WaitSetProperty_t** &prop)

  *<<**extension**>> (p. 432) Retrieves the **DDS_WaitSetProperty_t** (p. 1509) configuration of the associated **DDSWaitSet** (p. 1897).*
- virtual ~**DDSWaitSet** ()

  Destructor.
- **DDSWaitSet** ()

  Default no-argument constructor.
- **DDSWaitSet** (const **DDS_WaitSetProperty_t** &prop)

  *<<**extension**>> (p. 432) Constructor for a **DDSWaitSet** (p. 1897) that may delay for more while specifying that will be woken up after the given number of events or delay period, whichever happens first*

9.303.1 Detailed Description

*<<**interface**>> (p. 433)* Allows an application to wait until one or more of the attached **DDSCondition** (p. 1544) objects has a `trigger_value` of **DDS_BOOLEAN_TRUE** (p. 513) or else until the timeout expires.

9.303.2 Usage

DDSCondition (p. 1544) (s) (in conjunction with wait-sets) provide an alternative mechanism to allow the middleware to communicate communication status changes (including arrival of data) to the application.

This mechanism is wait-based. Its general use pattern is as follows:

- The application indicates which relevant information it wants to get by creating **DDSCondition** (p. 1544) objects (**DDSStatusCondition** (p. 1846), **DDSReadCondition** (p. 1842) or **DDSQueryCondition** (p. 1841)) and attaching them to a **DDSWaitSet** (p. 1897).

- It then waits on that **DDSWaitSet** (p. 1897) until the `trigger_value` of one or several **DDSCondition** (p. 1544) objects become **DDS_BOOLEAN_TRUE** (p. 513).

- It then uses the result of the wait (i.e., active_conditions, the list of **DDSCondition** (p. 1544) objects with `trigger_value` == **DDS_BOOLEAN_TRUE** (p. 513)) to actually get the information:

- by calling **DDSEntity::get_status_changes** (p. 1735) and then get_<communication_status>() on the relevant **DDSEntity** (p. 1731), if the condition is a **DDSStatusCondition** (p. 1846) and the status changes, refer to plain communication status;
- by calling **DDSEntity::get_status_changes** (p. 1735) and then **DDSSubscriber::get_datareaders** (p. 1874) on the relevant **DDSSubscriber** (p. 1860) (and then **FooDataReader::read()** (p. 1919) or **Foo↩ DataReader::take** (p. 1920) on the returned **DDSDataReader** (p. 1556) objects), if the condition is a **DDSStatusCondition** (p. 1846) and the status changes refers to **DDS_DATA_ON_READERS_STATUS** (p. 541);
- by calling **DDSEntity::get_status_changes** (p. 1735) and then **FooDataReader::read()** (p. 1919) or **Foo↩ DataReader::take** (p. 1920) on the relevant **DDSDataReader** (p. 1556), if the condition is a **DDSStatus↩ Condition** (p. 1846) and the status changes refers to **DDS_DATA_AVAILABLE_STATUS** (p. 541);
- by calling directly **FooDataReader::read_w_condition** (p. 1925) or **FooDataReader::take_w_condition** (p. 1926) on a **DDSDataReader** (p. 1556) with the **DDSCondition** (p. 1544) as a parameter if it is a **DDSReadCondition** (p. 1842) or a **DDSQueryCondition** (p. 1841).

Usually the first step is done in an initialization phase, while the others are put in the application main loop.

As there is no extra information passed from the middleware to the application when a wait returns (only the list of triggered **DDSCondition** (p. 1544) objects), **DDSCondition** (p. 1544) objects are meant to embed all that is needed to react properly when enabled. In particular, **DDSEntity** (p. 1731)-related conditions are related to exactly one **DDSEntity** (p. 1731) and cannot be shared.

The blocking behavior of the **DDSWaitSet** (p. 1897) is illustrated below.

The result of a **DDSWaitSet::wait** (p. 1901) operation depends on the state of the **DDSWaitSet** (p. 1897), which in turn depends on whether at least one attached **DDSCondition** (p. 1544) has a trigger_value of **DDS_BOOLEAN↩ _TRUE** (p. 513). If the wait operation is called on **DDSWaitSet** (p. 1897) with state BLOCKED, it will block the calling thread. If wait is called on a **DDSWaitSet** (p. 1897) with state UNBLOCKED, it will return immediately. In addition, when the **DDSWaitSet** (p. 1897) transitions from BLOCKED to UNBLOCKED it wakes up any threads that had called wait on it.

A key aspect of the Condition and WaitSet mechanism is the setting of the trigger_value of each **DDSCondition** (p. 1544).

The **DDSWaitSet** (p. 1897) cannot be used after calling **DDSDomainParticipantFactory::finalize_instance** (p. 1697).

9.303.3 Trigger State of a ::DDSStatusCondition

The trigger_value of a **DDSStatusCondition** (p. 1846) is the boolean OR of the ChangedStatusFlag of all the communication statuses (see **Status Kinds** (p. 533)) to which it is sensitive. That is, trigger_value == **DDS_↩ BOOLEAN_FALSE** (p. 513) only if all the values of the ChangedStatusFlags are **DDS_BOOLEAN_FALSE** (p. 513).

The sensitivity of the **DDSStatusCondition** (p. 1846) to a particular communication status is controlled by the list of enabled_statuses set on the condition by means of the **DDSStatusCondition::set_enabled_statuses** (p. 1847) operation.

Once the trigger_value of a StatusCondition becomes true, it remains true until the status that changed is reset. To reset a status, call the related get_*_status() operation. Or, in the case of the data available status, call read(), take(), or one of their variants. Therefore, if you are using a **DDSStatusCondition** (p. 1846) on a **DDSWaitSet** (p. 1897) to be notified of events, your thread will wake up when one of the statuses associated with the StatusCondition becomes true. If you do not reset the status, the StatusCondition trigger_value remains true and your WaitSet will not block again; it will immediately wake up when you call **DDSWaitSet::wait** (p. 1901).

9.303.4 Trigger State of a ::DDSReadCondition

Similar to the **DDSStatusCondition** (p. 1846), a **DDSReadCondition** (p. 1842) also has a `trigger_value` that determines whether the attached **DDSWaitSet** (p. 1897) is BLOCKED or UNBLOCKED. However, unlike the **DDSStatus↩ Condition** (p. 1846), the `trigger_value` of the **DDSReadCondition** (p. 1842) is tied to the presence of *at least a sample* managed by RTI Connext with **DDS_SampleStateKind** (p. 161) and **DDS_ViewStateKind** (p. 163) matching those of the **DDSReadCondition** (p. 1842). Furthermore, for the **DDSQueryCondition** (p. 1841) to have a `trigger↩ _value` == **DDS_BOOLEAN_TRUE** (p. 513), the data associated with the sample must be such that the `query_↩ expression` evaluates to **DDS_BOOLEAN_TRUE** (p. 513).

The fact that the trigger_value of a **DDSReadCondition** (p. 1842) depends on the presence of samples on the associated **DDSDataReader** (p. 1556) implies that a single take operation can potentially change the `trigger_↩ value` of several **DDSReadCondition** (p. 1842) or **DDSQueryCondition** (p. 1841) conditions. For example, if all samples are taken, any **DDSReadCondition** (p. 1842) and **DDSQueryCondition** (p. 1841) conditions associated with the **DDSDataReader** (p. 1556) that had their `trigger_value==TRUE` before will see the `trigger_value` change to FALSE. Note that this does not guarantee that **DDSWaitSet** (p. 1897) objects that were separately attached to those conditions will not be woken up. Once we have `trigger_value==TRUE` on a condition, it may wake up the attached **DDSWaitSet** (p. 1897), the condition transitioning to `trigger_value==FALSE` does not necessarily 'unwakeup' the WaitSet as 'unwakening' may not be possible in general.

The consequence is that an application blocked on a **DDSWaitSet** (p. 1897) may return from the wait with a list of conditions, some of which are not no longer 'active'. This is unavoidable if multiple threads are concurrently waiting on separate **DDSWaitSet** (p. 1897) objects and taking data associated with the same **DDSDataReader** (p. 1556) entity.

To elaborate further, consider the following example: A **DDSReadCondition** (p. 1842) that has a `sample_state↩ _mask` = {**DDS_NOT_READ_SAMPLE_STATE** (p. 162)} will have `trigger_value` of **DDS_BOOLEAN_TRUE** (p. 513) whenever a new sample arrives and will transition to **DDS_BOOLEAN_FALSE** (p. 513) as soon as all the newly-arrived samples are either read (so their sample state changes to READ) or taken (so they are no longer managed by RTI Connext). However if the same **DDSReadCondition** (p. 1842) had a `sample_state_mask` = { **DDS_READ↩ _SAMPLE_STATE** (p. 162), **DDS_NOT_READ_SAMPLE_STATE** (p. 162) }, then the `trigger_value` would only become **DDS_BOOLEAN_FALSE** (p. 513) once all the newly-arrived samples are taken (it is not sufficient to read them as that would only change the sample state to READ), which overlaps the mask on the **DDSReadCondition** (p. 1842).

9.303.5 Trigger State of a ::DDSGuardCondition

The `trigger_value` of a **DDSGuardCondition** (p. 1739) is completely controlled by the application via the operation **DDSGuardCondition::set_trigger_value** (p. 1741).

See also

Status Kinds (p. 533)

DDSStatusCondition (p. 1846), **DDSGuardCondition** (p. 1739)

DDSListener (p. 1794)

Examples

HelloWorld_subscriber.cxx.

9.303.6 Constructor & Destructor Documentation

9.303.6.1 ~DDSWaitSet()

```
virtual DDSWaitSet::~DDSWaitSet ( )  [virtual]
```

Destructor.

Releases the resources asociated with this **DDSWaitSet** (p. 1897).

Freeing a null pointer is safe and does nothing.

MT Safety:

UNSAFE. It is not safe to delete a **DDSWaitSet** (p. 1897) while another thread is calling an API that uses the entity. For instance, a thread must not delete a WaitSet while another thread is blocked with **DDSWaitSet::wait** (p. 1901). To properly handle this scenario, you can use a **DDSGuardCondition** (p. 1739) to wake up the WaitSet and then wait for the finalization of the thread.

9.303.6.2 DDSWaitSet() [1/2]

```
DDSWaitSet::DDSWaitSet ( )
```

Default no-argument constructor.

Construct a new **DDSWaitSet** (p. 1897).

9.303.6.3 DDSWaitSet() [2/2]

```
DDSWaitSet::DDSWaitSet (
            const  DDS_WaitSetProperty_t & prop )
```

<<***extension***>> *(p. 432)* Constructor for a **DDSWaitSet** (p. 1897) that may delay for more while specifying that will be woken up after the given number of events or delay period, whichever happens first

Constructs a new **DDSWaitSet** (p. 1897).

9.303.7 Member Function Documentation

9.303.7.1 wait()

```
virtual DDS_ReturnCode_t DDSWaitSet::wait (
            DDSConditionSeq & active_conditions,
          const DDS_Duration_t & timeout ) [virtual]
```

Allows an application thread to wait for the occurrence of certain conditions.

If none of the conditions attached to the **DDSWaitSet** (p. 1897) have a `trigger_value` of **DDS_BOOLEAN_TRUE** (p. 513), the wait operation will block, suspending the calling thread.

The result of the wait operation is the list of all the attached conditions that have a `trigger_value` of **DDS_↩ BOOLEAN_TRUE** (p. 513) (i.e., the conditions that unblocked the wait).

The wait operation takes a `timeout` argument that specifies the maximum duration for the wait. If this duration is exceeded and none of the attached **DDSCondition** (p. 1544) objects are **DDS_BOOLEAN_TRUE** (p. 513), wait fails with **DDS_RETCODE_TIMEOUT** (p. 533). In this case, the resulting list of conditions will be empty. If a negative duration is passed, wait fails with **DDS_RETCODE_BAD_PARAMETER** (p. 532).

Note: The resolution of the `timeout` period is constrained by the resolution of the system clock.

When the **DDSWaitSet** (p. 1897) is configured to wait for more than one trigger event and the timeout is exceeded before that number is reached, this function returns normally as long as at least one trigger event has occurred.

It is not allowable for more than one application thread to be waiting on the same **DDSWaitSet** (p. 1897). If the wait operation is invoked on a **DDSWaitSet** (p. 1897) that already has a thread blocking on it, the operation will return immediately with the value **DDS_RETCODE_PRECONDITION_NOT_MET** (p. 532).

Parameters

active_conditions	*<<**inout**>> (p. 433)* a valid non-NULL **DDSConditionSeq** (p. 1547) object. Note that RTI Connext will not allocate a new object if `active_conditions` is NULL; the method will return **DDS_RETCODE_PRECONDITION_NOT_MET** (p. 532).
timeout	*<<**in**>> (p. 433)* a wait timeout

Returns

One of the **Standard Return Codes** (p. 532) or **DDS_RETCODE_PRECONDITION_NOT_MET** (p. 532) or **DDS↩ _RETCODE_TIMEOUT** (p. 533).

Examples

HelloWorld_subscriber.cxx.

9.303.7.2 attach_condition()

```
virtual DDS_ReturnCode_t DDSWaitSet::attach_condition (
            DDSCondition * cond ) [virtual]
```

Attaches a **DDSCondition** (p. 1544) to the **DDSWaitSet** (p. 1897).

It is possible to attach a **DDSCondition** (p. 1544) on a **DDSWaitSet** (p. 1897) that is currently being waited upon (via the wait operation). In this case, if the **DDSCondition** (p. 1544) has a `trigger_value` of **DDS_BOOLEAN_TRUE** (p. 513), then attaching the condition will unblock the **DDSWaitSet** (p. 1897).

If the same **DDSCondition** (p. 1544) is attached multiple times, only the first attachment has any effect. Subsequent attempts to attach the same **DDSCondition** (p. 1544) are silently ignored. As a result, the **DDSCondition** (p. 1544) is only considered attached once and must only be detached once.

Parameters

cond	<<***in***>> *(p. 433)* Condition to be attached.

Returns

One of the **Standard Return Codes** (p. 532), or **DDS_RETCODE_OUT_OF_RESOURCES** (p. 533).

Examples

HelloWorld_subscriber.cxx.

9.303.7.3 detach_condition()

```
virtual  DDS_ReturnCode_t DDSWaitSet::detach_condition (
             DDSCondition * cond )  [virtual]
```

Detaches a **DDSCondition** (p. 1544) from the **DDSWaitSet** (p. 1897).

It is possible to detach a **DDSCondition** (p. 1544) on a **DDSWaitSet** (p. 1897) that is currently being waited upon (via the wait operation). If the **DDSCondition** (p. 1544) was not attached to the **DDSWaitSet** (p. 1897), the operation will return **DDS_RETCODE_BAD_PARAMETER** (p. 532).

If the same **DDSCondition** (p. 1544) had been previously attached multiple times, only a single detach is required; the **DDSCondition** (p. 1544) is considered attached only once.

Parameters

cond	<<***in***>> *(p. 433)* Condition to be detached.

Returns

One of the **Standard Return Codes** (p. 532), or **DDS_RETCODE_PRECONDITION_NOT_MET** (p. 532).

9.303.7.4 get_conditions()

```
virtual DDS_ReturnCode_t DDSWaitSet::get_conditions (
            DDSConditionSeq & attached_conditions ) [virtual]
```

Retrieves the list of attached **DDSCondition** (p. 1544) (s).

Parameters

attached_conditions	<<***inout***>> *(p. 433)* a **DDSConditionSeq** (p. 1547) object where the list of attached conditions will be returned

Returns

One of the **Standard Return Codes** (p. 532), or **DDS_RETCODE_OUT_OF_RESOURCES** (p. 533).

9.303.7.5 set_property()

```
virtual DDS_ReturnCode_t DDSWaitSet::set_property (
            const DDS_WaitSetProperty_t & prop ) [virtual]
```

<<***extension***>> *(p. 432)* Sets the **DDS_WaitSetProperty_t** (p. 1509), to configure the associated **DDSWaitSet** (p. 1897) to return after one or more trigger events have occurred.

Parameters

prop	<<***in***>> *(p. 433)*

Returns

One of the **Standard Return Codes** (p. 532)

9.303.7.6 get_property()

```
virtual DDS_ReturnCode_t DDSWaitSet::get_property (
            DDS_WaitSetProperty_t & prop ) [virtual]
```

<<***extension***>> *(p. 432)* Retrieves the **DDS_WaitSetProperty_t** (p. 1509) configuration of the associated **DDSWaitSet** (p. 1897).

Parameters

prop	<<***out***>> *(*p. *433)*

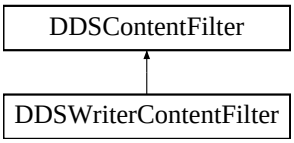
Returns

One of the **Standard Return Codes** (p. 532)

9.304 DDSWriterContentFilter Class Reference

<<***interface***>> *(*p. *433)* Interface to be used by a custom filter of a **DDSContentFilteredTopic** (p. 1551).

Inheritance diagram for DDSWriterContentFilter:



Public Member Functions

- virtual **DDS_ReturnCode_t writer_compile** (void ∗writer_filter_data, **DDS_ExpressionProperty** ∗prop, const char ∗expression, const **DDS_StringSeq** ∗parameters, const **DDS_TypeCode** ∗type_code, const char ∗type↩ _class_name, const **DDS_Cookie_t** ∗cookie)=0

  A writer-side filtering API to compile an instance of the content filter according to the filter expression and parameters specified by a matching ***DDSDataReader*** *(*p. *1556).*
- virtual struct **DDS_CookieSeq** ∗ **writer_evaluate** (void ∗writer_filter_data, const void ∗sample, const **DDS_↩ FilterSampleInfo** ∗meta_data)=0

  A writer-side filtering API to retrieve a list of DataReaders whose content filters pass the sample.
- virtual void **writer_finalize** (void ∗writer_filter_data, const **DDS_Cookie_t** ∗cookie)=0

  A writer-side filtering API to clean up a previously compiled instance of the content filter.
- virtual **DDS_ReturnCode_t writer_attach** (void ∗∗writer_filter_data)=0

  A writer-side filtering API to create some state that can facilitate filtering on the writer side.
- virtual void **writer_detach** (void ∗writer_filter_data)=0

  A writer-side filtering API to clean up a previously created state using ***DDSWriterContentFilter::writer_attach*** *(*p. *1908).*
- virtual void **writer_return_loan** (void ∗writer_filter_data, **DDS_CookieSeq** ∗cookies)=0

  A writer-side filtering API to return the loan on the list of DataReaders returned by ***DDSWriterContentFilter::writer_↩ evaluate*** *(*p. *1907).*

9.304.1 Detailed Description

<<***interface***>> *(p. 433)* Interface to be used by a custom filter of a **DDSContentFilteredTopic** (p. 1551).

Entity:

DDSContentFilteredTopic (p. 1551)

This interface can be implemented by an application-provided class and then registered with the **DDSDomain↩ Participant** (p. 1619) such that samples can be filtered for a **DDSContentFilteredTopic** (p. 1551) with the given filter name.

Note: The API for using a custom content filter is subject to change in a future release.

See also

DDSContentFilteredTopic (p. 1551)

DDSDomainParticipant::register_contentfilter (p. 1632)

9.304.2 Member Function Documentation

9.304.2.1 writer_compile()

```
virtual  DDS_ReturnCode_t DDSWriterContentFilter::writer_compile (
             void * writer_filter_data,
              DDS_ExpressionProperty * prop,
             const char * expression,
             const  DDS_StringSeq * parameters,
             const  DDS_TypeCode * type_code,
             const char * type_class_name,
             const  DDS_Cookie_t * cookie )  [pure virtual]
```

A writer-side filtering API to compile an instance of the content filter according to the filter expression and parameters specified by a matching **DDSDataReader** (p. 1556).

This method is called when the **DDSDataWriter** (p. 1589) discovers a **DDSDataReader** (p. 1556) with a **DDSContent↩ FilteredTopic** (p. 1551) or when a **DDSDataWriter** (p. 1589) is notified of a change in a DataReader's filter parameter for the locally registered content filter instance.

It is possible for multiple threads to be calling into this function at the same time.

Parameters

writer_filter_data	<<***in***>> *(p. 433)* A pointer to the state created using **DDSWriterContentFilter::writer_attach** (p. 1908) .
prop	<<***out***>> *(p. 433)* A pointer to **DDS_ExpressionProperty** (p. 1152) that allows you to indicate to RTI Connext if a filter expression can be optimized.

Parameters

expression	*<<**in**>> (p. 433)* An ASCIIZ string with the filter expression. The memory used by this string is owned by RTI Connext and **must** not be freed. If you want to manipulate this string, you **must** first make a copy of it.
parameters	*<<**in**>> (p. 433)* A string sequence with the expression parameters with which the **DDSContentFilteredTopic** (p. 1551) was created. The string sequence is **equal** (but **not** identical) to the string sequence passed to **DDSDomainParticipant::create_contentfilteredtopic** (p. 1653). Note that the sequence passed to the compile function is **owned** by RTI Connext and **must not** be referenced outside the compile function.
type_code	*<<**in**>> (p. 433)* A pointer to the type code for the related **DDSTopic** (p. 1885) of the **DDSContentFilteredTopic** (p. 1551). A type_code is a description of a type in terms of which types it contains (int32, string, etc.) and the corresponding member field names in the data type structure. The type code can be used to write custom content filters that can be used with any type.
type_class_name	*<<**in**>> (p. 433)* Fully qualified class name of the related **DDSTopic** (p. 1885).
cookie	*<<**in**>> (p. 433)* **DDS_Cookie_t** (p. 830) to uniquely identify **DDSDataReader** (p. 1556) for which **DDSWriterContentFilter::writer_compile** (p. 1906) was called.

Exceptions

One	of the **Standard Return Codes** (p. 532)

9.304.2.2 writer_evaluate()

```
virtual struct  DDS_CookieSeq * DDSWriterContentFilter::writer_evaluate (
             void * writer_filter_data,
             const void * sample,
             const  DDS_FilterSampleInfo * meta_data )  [pure virtual]
```

A writer-side filtering API to retrieve a list of DataReaders whose content filters pass the sample.

This method is called every time a **DDSDataWriter** (p. 1589) writes a new sample. Its purpose is to evaluate the sample for all the readers for which the **DDSDataWriter** (p. 1589) is performing writer-side filtering and return the list of **DDS↩ _Cookie_t** (p. 830) structures associated with the DataReaders whose filters pass the sample.

It is possible for multiple threads to be calling into this function at the same time

Parameters

writer_filter_data	*<<**in**>> (p. 433)* A pointer to the state created using **DDSWriterContentFilter::writer_attach** (p. 1908) .
sample	*<<**in**>> (p. 433)* Pointer to a deserialized sample to be filtered.
meta_data	*<<**in**>> (p. 433)* Pointer to meta data associated with the sample.

Returns

The function returns **DDS_CookieSeq** (p. 830) which identifies the set of DataReaders whose filters pass the sample.

9.304.2.3 writer_finalize()

```
virtual void DDSWriterContentFilter::writer_finalize (
            void * writer_filter_data,
            const  DDS_Cookie_t * cookie )  [pure virtual]
```

A writer-side filtering API to clean up a previously compiled instance of the content filter.

This method is called to notify the filter implementation that the **DDSDataWriter** (p. 1589) is no longer matching with a **DDSDataReader** (p. 1556) for which it was previously performing writer-side filtering. This will allow the filter to purge any state it was maintaining for the **DDSDataReader** (p. 1556).

It is possible for multiple threads to be calling into this function at the same time.

Parameters

writer_filter_data	<<***in***>> *(p. 433)* A pointer to the state created using **DDSWriterContentFilter::writer_attach** (p. 1908).
cookie	**DDS_Cookie_t** (p. 830) to uniquely identify **DDSDataReader** (p. 1556) for which **DDSWriterContentFilter::writer_finalize** (p. 1908) was called.

9.304.2.4 writer_attach()

```
virtual  DDS_ReturnCode_t DDSWriterContentFilter::writer_attach (
            void ** writer_filter_data )  [pure virtual]
```

A writer-side filtering API to create some state that can facilitate filtering on the writer side.

This method is called to create some state required to perform filtering on the writer side using writer-side filtering APIs. This method will be called for every **DDSDataWriter** (p. 1589); it will be called only the first time the **DDSData↩ Writer** (p. 1589) matches a **DDSDataReader** (p. 1556) using the specified filter. This function will not be called for any subsequent DataReaders that match the DataWriter and are using the same filter.

Parameters

writer_filter_data	<<***out***>> *(p. 433)* A user-specified opaque pointer to some state created on the **DDSDataWriter** (p. 1589) that will help perform writer-side filtering efficiently.

9.304.2.5 writer_detach()

```
virtual void DDSWriterContentFilter::writer_detach (
            void * writer_filter_data )  [pure virtual]
```

A writer-side filtering API to clean up a previously created state using **DDSWriterContentFilter::writer_attach** (p. 1908).

This method is called to delete any state created using the **DDSWriterContentFilter::writer_attach** (p. 1908) function. This method will be called when the **DDSDataWriter** (p. 1589) is deleted.

Parameters

writer_filter_data	<<***in***>> *(p. 433)* A pointer to the state created using **DDSWriterContentFilter::writer_attach** (p. 1908).

9.304.2.6 writer_return_loan()

```
virtual void DDSWriterContentFilter::writer_return_loan (
            void * writer_filter_data,
             DDS_CookieSeq * cookies )  [pure virtual]
```

A writer-side filtering API to return the loan on the list of DataReaders returned by **DDSWriterContentFilter::writer_↩ evaluate** (p. 1907).

This method is called to return the loan on **DDS_CookieSeq** (p. 830) returned by **DDSWriterContentFilter::writer_↩ return_loan** (p. 1909). It is possible for multiple threads to be calling into this function at the same time.

Parameters

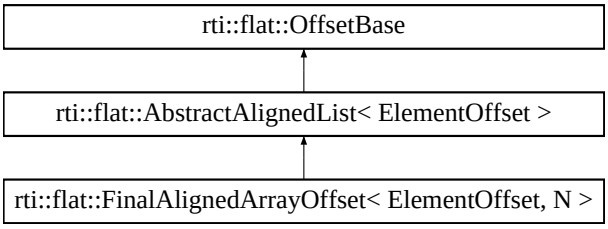
writer_filter_data	<<***in***>> *(p. 433)* A pointer to the state created using **DDSWriterContentFilter::writer_attach** (p. 1908).
cookies	<<***in***>> *(p. 433)* **DDS_CookieSeq** (p. 830) for which the **DDSWriterContentFilter::writer_return_loan** (p. 1909). was invoked.

9.305 rti::flat::FinalAlignedArrayOffset< ElementOffset, N > Class Template Reference

Offset to an array of final elements.

```
#include <SequenceOffsets.hpp>
```

Inheritance diagram for rti::flat::FinalAlignedArrayOffset< ElementOffset, N >:



Public Member Functions

- ElementOffset **get_element** (unsigned int i)

  Gets the Offset to an element.

Additional Inherited Members

9.305.1 Detailed Description

template<typename ElementOffset, unsigned int N>
class rti::flat::FinalAlignedArrayOffset< ElementOffset, N >

Offset to an array of final elements.

Template Parameters

ElementOffset	A final struct Offset, such as **MyFlatFinalOffset** (p. 2014).
N	The array bound. For multidimensional arrays, N is the product of each dimension bound.

Represents an Offset to an array member and allows getting an Offset to each of its elements.

The following code shows how to use a **FinalAlignedArrayOffset** (p. 1909) to initialize an array member with **MyFlat↩ MutableBuilder** (p. 2018):

```
MyFlatMutableBuilder builder = rti::flat::build_data(writer);
auto array_offset = builder.add_my_final_array();
for (MyFlatFinalOffset element_offset : array_offset) {
    element_offset.my_primitive(3);
    // ...
}
// Or access an element directly:
auto element_offset = array_offset.get_element(3);
element_offset.my_primitive(3);
```

A more efficient way to access a final array, provided it complies with the required preconditions, is through **rti::flat↩ ::plain_cast()** (p. 773).

FinalArrayOffset (p. 1911) and **FinalAlignedArrayOffset** (p. 1909) provide the same interface, but have different implementation details. **FinalArrayOffset** (p. 1911) is used when the array member is part of a final type too, whereas **FinalAlignedArrayOffset** (p. 1909) corresponds to an array inside a mutable type.

A **FinalAlignedArrayOffset** (p. 1909) may meet the requirements to be cast to an array of the equivalent plain C++ element type (see **rti::flat::plain_cast()** (p. 773)).

See also

MutableArrayOffset (p. 2010) encapsulates arrays of variable-size elements

9.305.2 Member Function Documentation

9.305.2.1 get_element()

```
template<typename ElementOffset , unsigned int N>
ElementOffset  rti::flat::FinalAlignedArrayOffset< ElementOffset, N >::get_element (
             unsigned int i )  [inline]
```

Gets the Offset to an element.

Parameters

i	The zero-based index to the element

Returns

The Offset to the element in the i-th position

See also

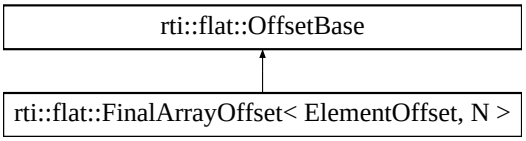
rti::flat::plain_cast() (p. 773) for a method to access all the elements at once

9.306 rti::flat::FinalArrayOffset< ElementOffset, N > Class Template Reference

Offset to an array of final elements.

```
#include <SequenceOffsets.hpp>
```

Inheritance diagram for rti::flat::FinalArrayOffset< ElementOffset, N >:



Public Member Functions

- ElementOffset **get_element** (unsigned int i)

  Gets the Offset to an element.

9.306.1 Detailed Description

template<typename ElementOffset, unsigned int N>
class rti::flat::FinalArrayOffset< ElementOffset, N >

Offset to an array of final elements.

Template Parameters

ElementOffset	A final struct Offset, such as **MyFlatFinalOffset** (p. 2014).
N	The array bound. For multidimensional arrays, N is the product of each dimension bound.

Represents an Offset to an array member and allows getting an Offset to each of its elements.

FinalArrayOffset (p. 1911) and **FinalAlignedArrayOffset** (p. 1909) provide the same interface, but have different implementation details. **FinalArrayOffset** (p. 1911) is used when the array member is part of a final type too, whereas **FinalAlignedArrayOffset** (p. 1909) corresponds to an array inside a mutable type.

A **FinalArrayOffset** (p. 1911) may meet the requirements to be cast to an array of the equivalent plain C++ element type (see **rti::flat::plain_cast()** (p. 773)).

9.306.2 Member Function Documentation

9.306.2.1 get_element()

```
template<typename ElementOffset , unsigned int N>
ElementOffset  rti::flat::FinalArrayOffset< ElementOffset, N >::get_element (
             unsigned int i )  [inline]
```

Gets the Offset to an element.

Parameters

i	The zero-based index to the element

Returns

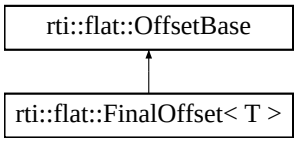
The Offset to the element in the i-th position

9.307 rti::flat::FinalOffset< T > Class Template Reference

The base class of all Offsets to a final struct type.

`#include <Offset.hpp>`

Inheritance diagram for rti::flat::FinalOffset< T >:



Additional Inherited Members

9.307.1 Detailed Description

template<typename T>
class rti::flat::FinalOffset< T >

The base class of all Offsets to a final struct type.

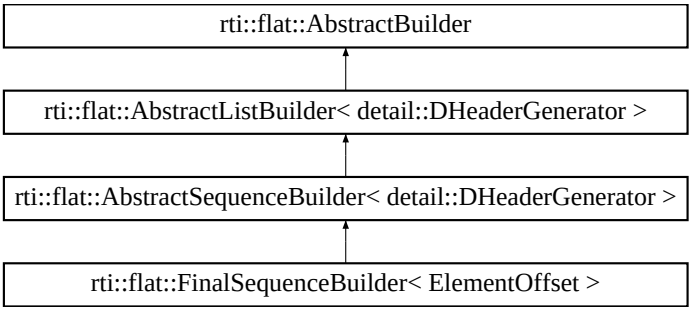
This class contains only implementation details; all the public accessors are defined in the generated type (**MyFlat↩ FinalOffset** (p. 2014)).

9.308 rti::flat::FinalSequenceBuilder< ElementOffset > Class Template Reference

Builds a sequence member of fixed-size elements.

`#include <SequenceBuilders.hpp>`

Inheritance diagram for rti::flat::FinalSequenceBuilder< ElementOffset >:



Public Member Functions

- ElementOffset **add_next** ()

  Adds the next element.
- **FinalSequenceBuilder** & **add_n** (unsigned int count)

  Adds a number of elements at once.
- **Offset finish** ()

  Finishes building the sequence.

Additional Inherited Members

9.308.1 Detailed Description

template<typename ElementOffset>
class rti::flat::FinalSequenceBuilder< ElementOffset >

Builds a sequence member of fixed-size elements.

Template Parameters

ElementOffset	The Offset type for the elements of the sequence

To add an element, call **add_next()** (p. 1914) and use the ElementOffset it returns to initialize the element's values. An empty sequence can be built by calling **finish()** (p. 1915) without any call to **add_next()** (p. 1914).

This class doesn't enforce the sequence bound set in IDL.

The following example uses a **FinalSequenceBuilder** (p. 1913) to initialize a sequence member of **MyFlatMutable↩ Builder** (p. 2018) with two elements:

```
MyFlatMutableBuilder builder = rti::flat::build_data(writer);
auto seq_builder = builder.build_my_final_seq();
MyFlatFinalOffset element = seq_builder.add_next();
element.my_primitive(1);
// ... continue initializing the first element
element = seq_builder.add_next();
element.my_primitive(2);
// ... continue initializing the second element

seq_builder.finish();
```

If the element type meets certain requirements, **rti::flat::plain_cast()** (p. 773) provides a more efficient way to initialize a sequence of final elements.

9.308.2 Member Function Documentation

9.308.2.1 add_next()

```
template<typename ElementOffset >
ElementOffset  rti::flat::FinalSequenceBuilder< ElementOffset >::add_next ( )  [inline]
```

Adds the next element.

Returns

The Offset that can be used to set the element values

9.308.2.2 add_n()

```
template<typename ElementOffset >
FinalSequenceBuilder &  rti::flat::FinalSequenceBuilder< ElementOffset >::add_n (
            unsigned int count )  [inline]
```

Adds a number of elements at once.

This is an alternative to **add_next()** (p. 1914).

To initialize the elements, call **finish()** (p. 1915) and use the Offset it returns to access the elements.

9.308.2.3 finish()

```
template<typename ElementOffset >
Offset  rti::flat::FinalSequenceBuilder< ElementOffset >::finish ( )  [inline]
```

Finishes building the sequence.

Returns

An Offset to the member that has been built.

See also

discard() (p. 788)

9.309 rti::flat::flat_type_traits< T > Struct Template Reference

Given a **Sample** (p. 2187), an Offset or a Builder, it allows obtaining the other types.

9.309.1 Detailed Description

template<typename T>
struct rti::flat::flat_type_traits< T >

Given a **Sample** (p. 2187), an Offset or a Builder, it allows obtaining the other types.

Template Parameters

T	One of the following: • A **Sample** (p. 2187) type, such as **MyFlatMutable** (p. 770) • An Offset type, such as **MyFlatMutableOffset** (p. 2025) • A Builder type, such as **MyFlatMutableBuilder** (p. 2018)

Given `T`, this type provides the following typedefs:

- `flat_type_traits<T>::offset`, T's related offset type (undefined if T itself is an Offset)
- `flat_type_traits<T>::builder`, T's related builder type (undefined if T itself is a Builder, or the topic-type is not mutable)
- `flat_type_traits<T>::flat_type`, T's related **Sample** (p. 2187) type (undefined if T itself is a **Sample** (p. 2187) type)
- `flat_type_traits<T>::plain_type`, T's equivalent definition as a plain (non-FlatData) type.

For example, for `T` = **MyFlatMutable** (p. 770), **flat_type_traits** (p. 1915) is defined as follows:

```
template <>
 struct flat_type_traits<MyFlatMutable> {
     typedef MyFlatMutablePlainHelper plain_type;
     typedef MyFlatMutableOffset offset;
     typedef MyFlatMutableBuilder builder;
 };
```

Or if `T` = **MyFlatMutableOffset** (p. 2025):

```
template <>
struct flat_type_traits<MyFlatMutableOffset> {
      typedef MyFlatMutable flat_type;
      typedef MyFlatMutablePlainHelper plain_type;
      typedef MyFlatMutableBuilder builder;
};
```

See also

rti::flat::plain_cast() (p. 773)

9.310 Foo Struct Reference

A representative user-defined data type.

9.310.1 Detailed Description

A representative user-defined data type.

Foo (p. 1916) represents a user-defined data-type that is intended to be distributed using DDS.

The type **Foo** (p. 1916) is usually defined using IDL syntax and placed in a ".idl" file that is then processed using rtiddsgen. The rtiddsgen utility generates the helper classes **FooSeq** (p. 1965) as well as the necessary code for DDS to manipulate the type (serialize it so that it can be sent over the network) as well as the implied **FooDataReader** (p. 1917) and **FooDataWriter** (p. 1944) types that allow the application to send and receive data of this type.

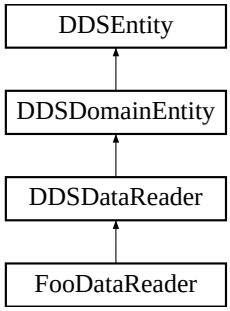
See also

FooSeq (p. 1965), **FooDataWriter** (p. 1944), **FooDataReader** (p. 1917), **FooTypeSupport** (p. 1979), the `Code Generator User's Manual`

9.311 FooDataReader Class Reference

<<***interface***>> *(*p. *433)* <<***generic***>> *(*p. *433)* User data type-specific data reader.

Inheritance diagram for FooDataReader:



Public Member Functions

- virtual **DDS_ReturnCode_t read** (**FooSeq** &received_data, **DDS_SampleInfoSeq** &info_seq, **DDS_Long** max_samples, **DDS_SampleStateMask** sample_states, **DDS_ViewStateMask** view_states, **DDS_Instance**↩ **StateMask** instance_states)=0

  *Access a collection of data samples from the **DDSDataReader** (*p. *1556).*
- virtual **DDS_ReturnCode_t take** (**FooSeq** &received_data, **DDS_SampleInfoSeq** &info_seq, **DDS_Long** max_samples, **DDS_SampleStateMask** sample_states, **DDS_ViewStateMask** view_states, **DDS_Instance**↩ **StateMask** instance_states)=0

  *Access a collection of data-samples from the **DDSDataReader** (*p. *1556).*
- virtual **DDS_ReturnCode_t read_w_condition** (**FooSeq** &received_data, **DDS_SampleInfoSeq** &info_seq, **DDS_Long** max_samples, **DDSReadCondition** ∗condition)=0

  *Accesses via **FooDataReader::read** (*p. *1919) the samples that match the criteria specified in the **DDSReadCondition** (*p. *1842).*
- virtual **DDS_ReturnCode_t take_w_condition** (**FooSeq** &received_data, **DDS_SampleInfoSeq** &info_seq, **DDS_Long** max_samples, **DDSReadCondition** ∗condition)=0

  *Analogous to **FooDataReader::read_w_condition** (*p. *1925) except it accesses samples via the **FooDataReader::take** (*p. *1920) operation.*
- virtual **DDS_ReturnCode_t read_next_sample** (**Foo** &received_data, **DDS_SampleInfo** &sample_info)=0

  *Copies the next not-previously-accessed data value from the **DDSDataReader** (*p. *1556).*
- virtual **DDS_ReturnCode_t take_next_sample** (**Foo** &received_data, **DDS_SampleInfo** &sample_info)=0

  *Copies the next not-previously-accessed data value from the **DDSDataReader** (*p. *1556).*
- virtual **DDS_ReturnCode_t read_instance** (**FooSeq** &received_data, **DDS_SampleInfoSeq** &info_seq, **DDS_Long** max_samples, const **DDS_InstanceHandle_t** &a_handle, **DDS_SampleStateMask** sample_states, **DDS_ViewStateMask** view_states, **DDS_InstanceStateMask** instance_states)=0

  *Access a collection of data samples from the **DDSDataReader** (*p. *1556).*
- virtual **DDS_ReturnCode_t take_instance** (**FooSeq** &received_data, **DDS_SampleInfoSeq** &info_seq, **DDS_Long** max_samples, const **DDS_InstanceHandle_t** &a_handle, **DDS_SampleStateMask** sample_states, **DDS_ViewStateMask** view_states, **DDS_InstanceStateMask** instance_states)=0

  *Access a collection of data samples from the **DDSDataReader** (*p. *1556).*
- virtual **DDS_ReturnCode_t read_instance_w_condition** (**FooSeq** &received_data, **DDS_SampleInfoSeq** &info_seq, **DDS_Long** max_samples, const **DDS_InstanceHandle_t** &previous_handle, **DDSReadCondition** ∗condition)=0

*<<**extension**>> (p. 432) Accesses via **FooDataReader::read_instance** (p. 1929) the samples that match the criteria specified in the **DDSReadCondition** (p. 1842).*

- virtual **DDS_ReturnCode_t take_instance_w_condition** (**FooSeq** &received_data, **DDS_SampleInfoSeq** &info_seq, **DDS_Long** max_samples, const **DDS_InstanceHandle_t** &previous_handle, **DDSReadCondition** *condition)=0

  *<<**extension**>> (p. 432) Accesses via **FooDataReader::take_instance** (p. 1931) the samples that match the criteria specified in the **DDSReadCondition** (p. 1842).*
- virtual **DDS_ReturnCode_t read_next_instance** (**FooSeq** &received_data, **DDS_SampleInfoSeq** &info_↩ seq, **DDS_Long** max_samples, const **DDS_InstanceHandle_t** &previous_handle, **DDS_SampleStateMask** sample_states, **DDS_ViewStateMask** view_states, **DDS_InstanceStateMask** instance_states)=0

  *Access a collection of data samples from the **DDSDataReader** (p. 1556).*
- virtual **DDS_ReturnCode_t take_next_instance** (**FooSeq** &received_data, **DDS_SampleInfoSeq** &info_↩ seq, **DDS_Long** max_samples, const **DDS_InstanceHandle_t** &previous_handle, **DDS_SampleStateMask** sample_states, **DDS_ViewStateMask** view_states, **DDS_InstanceStateMask** instance_states)=0

  *Access a collection of data samples from the **DDSDataReader** (p. 1556).*
- virtual **DDS_ReturnCode_t read_next_instance_w_condition** (**FooSeq** &received_data, **DDS_Sample**↩ **InfoSeq** &info_seq, **DDS_Long** max_samples, const **DDS_InstanceHandle_t** &previous_handle, **DDSRead**↩ **Condition** *condition)=0

  *Accesses via **FooDataReader::read_next_instance** (p. 1935) the samples that match the criteria specified in the **DDSReadCondition** (p. 1842).*
- virtual **DDS_ReturnCode_t take_next_instance_w_condition** (**FooSeq** &received_data, **DDS_Sample**↩ **InfoSeq** &info_seq, **DDS_Long** max_samples, const **DDS_InstanceHandle_t** &previous_handle, **DDSRead**↩ **Condition** *condition)=0

  *Accesses via **FooDataReader::take_next_instance** (p. 1936) the samples that match the criteria specified in the **DDSReadCondition** (p. 1842).*
- virtual **DDS_ReturnCode_t return_loan** (**FooSeq** &received_data, **DDS_SampleInfoSeq** &info_seq)=0

  *Indicates to the **DDSDataReader** (p. 1556) that the application is done accessing the collection of `received_data` and `info_seq` obtained by some earlier invocation of read or take on the **DDSDataReader** (p. 1556).*
- virtual **DDS_ReturnCode_t get_key_value** (**Foo** &key_holder, const **DDS_InstanceHandle_t** handle)=0

  Retrieve the instance `key` that corresponds to an instance `handle`.
- virtual **DDS_InstanceHandle_t lookup_instance** (const **Foo** &key_holder)=0

  Retrieves the instance `handle` that corresponds to an instance `key_holder`.
- virtual **DDS_ReturnCode_t is_data_consistent** (**DDS_Boolean** &is_data_consistent, const TData *sample, const **DDS_SampleInfo** *sample_info)=0

  When using Zero Copy transfer over shared memory, checks if the sample has been overwritten by the DataWriter.

Static Public Member Functions

- static **FooDataReader** * **narrow** (**DDSDataReader** *reader)

  *Narrow the given **DDSDataReader** (p. 1556) pointer to a **FooDataReader** (p. 1917) pointer.*

9.311.1 Detailed Description

*<<**interface**>> (p. 433) <<**generic**>> (p. 433)* User data type-specific data reader.

Defines the user data type specific reader interface generated for each application class.

The concrete user data type reader automatically generated by the implementation is an incarnation of this class.

See also

DDSDataReader (p. 1556)

Foo (p. 1916)

FooDataWriter (p. 1944)

the Code Generator User's Manual

9.311.2 Member Function Documentation

9.311.2.1 narrow()

```
static FooDataReader * FooDataReader::narrow (
            DDSDataReader * reader ) [static]
```

Narrow the given **DDSDataReader** (p. 1556) pointer to a **FooDataReader** (p. 1917) pointer.

9.311.2.2 read()

```
virtual DDS_ReturnCode_t FooDataReader::read (
            FooSeq & received_data,
            DDS_SampleInfoSeq & info_seq,
            DDS_Long max_samples,
            DDS_SampleStateMask sample_states,
            DDS_ViewStateMask view_states,
            DDS_InstanceStateMask instance_states ) [pure virtual]
```

Access a collection of data samples from the **DDSDataReader** (p. 1556).

This operation offers the same functionality and API as **FooDataReader::take** (p. 1920) except that the samples returned remain in the **DDSDataReader** (p. 1556) such that they can be retrieved again by means of a read or take operation.

Please refer to the documentation of **FooDataReader::take()** (p. 1920) for details on the number of samples returned within the received_data and info_seq as well as the order in which the samples appear in these sequences.

The act of reading a sample changes its sample_state to **DDS_READ_SAMPLE_STATE** (p. 162). If the sample belongs to the most recent generation of the instance, it will also set the view_state of the instance to be **DDS_↩ NOT_NEW_VIEW_STATE** (p. 164). It will not affect the instance_state of the instance.

Once the application completes its use of the samples, it must 'return the loan' to the **DDSDataReader** (p. 1556) by calling the **FooDataReader::return_loan** (p. 1940) operation.

Important: When you loan data from the middleware, you *must not* keep any pointers to any part of the data samples or the **DDS_SampleInfo** (p. 1344) objects after the call to **FooDataReader::return_loan** (p. 1940). Returning the loan places the objects back into a pool, allowing the middleware to overwrite them with new data.

Note: While the loan must be returned at some point, you do *not* have to do so before the next **FooDataReader::take** (p. 1920) call. However, failure to return the loan will eventually deplete the **DDSDataReader** (p. 1556) of the buffers it needs to receive new samples and eventually samples will start to be lost. The total number of buffers available to the **DDSDataReader** (p. 1556) is specified by the **DDS_ResourceLimitsQosPolicy** (p. 1308) and the **DDS_DataReader↩ ResourceLimitsQosPolicy** (p. 867).

Important: If the samples "returned" by this method are loaned from RTI Connext (see **FooDataReader::take** (p. 1920) for more information on memory loaning), it is important that their contents not be changed. Because the memory in which the data is stored belongs to the middleware, any modifications made to the data will be seen the next time the same samples are read or taken; the samples will no longer reflect the state that was received from the network.

Parameters

received_data	<<***inout***>> *(*p. *433)* User data type-specific **FooSeq** (p. 1965) object where the received data samples will be returned.
info_seq	<<***inout***>> *(*p. *433)* A **DDS_SampleInfoSeq** (p. 1355) object where the received sample info will be returned.
max_samples	<<***in***>> *(*p. *433)* The maximum number of samples to be returned. If the special value **DDS_LENGTH_UNLIMITED** (p. 636) is provided, as many samples will be returned as are available, up to the limits described in the documentation for **FooDataReader::take()** (p. 1920).
sample_states	<<***in***>> *(*p. *433)* Data samples matching one of these `sample_states` are returned.
view_states	<<***in***>> *(*p. *433)* Data samples matching one of these `view_state` are returned.
instance_states	<<***in***>> *(*p. *433)* Data samples matching ones of these `instance_state` are returned.

Exceptions

One	of the **Standard Return Codes** (p. 532), **DDS_RETCODE_PRECONDITION_NOT_MET** (p. 532), **DDS_RETCODE_NO_DATA** (p. 533) or **DDS_RETCODE_NOT_ENABLED** (p. 533).

See also

FooDataReader::take (p. 1920)

FooDataReader::read_w_condition (p. 1925),

FooDataReader::take_w_condition (p. 1926)

DDS_LENGTH_UNLIMITED (p. 636)

9.311.2.3 take()

```
virtual  DDS_ReturnCode_t FooDataReader::take (
              FooSeq & received_data,
              DDS_SampleInfoSeq & info_seq,
              DDS_Long max_samples,
              DDS_SampleStateMask sample_states,
              DDS_ViewStateMask view_states,
              DDS_InstanceStateMask instance_states )  [pure virtual]
```

Access a collection of data-samples from the **DDSDataReader** (p. 1556).

The operation will return the list of samples received by the **DDSDataReader** (p. 1556) since the last **FooDataReader::take** (p. 1920) operation that matches the specified **DDS_SampleStateMask** (p. 161), **DDS_ViewStateMask** (p. 163) and **DDS_InstanceStateMask** (p. 165).

This operation may fail with **DDS_RETCODE_ERROR** (p. 532) if the **DDS_DataReaderResourceLimitsQosPolicy::max_outstanding_reads** (p. 873) limit has been exceeded.

The actual number of samples returned depends on the information that has been received by the middleware as well as the **DDS_HistoryQosPolicy** (p. 1162), **DDS_ResourceLimitsQosPolicy** (p. 1308), **DDS_DataReaderResourceLimitsQosPolicy** (p. 867) and the characteristics of the data-type that is associated with the **DDSDataReader** (p. 1556):

- In the case where the **DDS_HistoryQosPolicy::kind** (p. 1164) is **DDS_KEEP_LAST_HISTORY_QOS** (p. 604), the call will return at most **DDS_HistoryQosPolicy::depth** (p. 1164) samples for each ALIVE instance and (**DDS_HistoryQosPolicy::depth** (p. 1164) +1) samples for each NOT_ALIVE instance. The extra sample is an invalid sample (**DDS_SampleInfo::valid_data** (p. 1351) is FALSE) that is used to indicate the instance state transition from ALIVE to NOT_ALIVE.

- The maximum number of samples returned is limited by **DDS_ResourceLimitsQosPolicy::max_samples** (p. 1311), and by **DDS_DataReaderResourceLimitsQosPolicy::max_samples_per_read** (p. 873).

- For multiple instances, the number of samples returned is additionally limited by the product (**DDS_ResourceLimitsQosPolicy::max_samples_per_instance** (p. 1311)
  - **DDS_ResourceLimitsQosPolicy::max_instances** (p. 1311))

- If **DDS_DataReaderResourceLimitsQosPolicy::max_infos** (p. 871) is limited, the number of samples returned may also be limited if insufficient **DDS_SampleInfo** (p. 1344) resources are available.

If the read or take succeeds and the number of samples returned has been limited (by means of a maximum limit, as listed above, or insufficient **DDS_SampleInfo** (p. 1344) resources), the call will complete successfully and provide those samples the reader is able to return. The user may need to make additional calls, or return outstanding loaned buffers in the case of insufficient resources, in order to access remaining samples.

Note that in the case where the **DDSTopic** (p. 1885) associated with the **DDSDataReader** (p. 1556) is bound to a data-type that has no key definition, then there will be at most one instance in the **DDSDataReader** (p. 1556). So the per-sample limits will apply.

The act of *taking* a sample removes it from RTI Connext so it cannot be read or taken again. If the sample belongs to the most recent generation of the instance, it will also set the `view_state` of the sample's instance to **DDS_NOT_NEW_VIEW_STATE** (p. 164). It will not affect the `instance_state` of the sample's instance.

After **FooDataReader::take** (p. 1920) completes, `received_data` and `info_seq` will be of the same length and contain the received data.

If the sequences are empty (maximum size equals 0) when the **FooDataReader::take** (p. 1920) is called, the samples returned in the `received_data` and the corresponding `info_seq` are 'loaned' to the application from buffers provided by the **DDSDataReader** (p. 1556). The application can use them as desired and has guaranteed exclusive access to them.

Once the application completes its use of the samples it must 'return the loan' to the **DDSDataReader** (p. 1556) by calling the **FooDataReader::return_loan** (p. 1940) operation.

Important: When you loan data from the middleware, you *must not* keep any pointers to any part of the data samples or the **DDS_SampleInfo** (p. 1344) objects after the call to **FooDataReader::return_loan** (p. 1940). Returning the loan places the objects back into a pool, allowing the middleware to overwrite them with new data.

Note: While you must call **FooDataReader::return_loan** (p. 1940) at some point, you do *not* have to do so before the next **FooDataReader::take** (p. 1920) call. However, failure to return the loan will eventually deplete the **DDSDataReader** (p. 1556) of the buffers it needs to receive new samples and eventually samples will start to be lost. The total number of buffers available to the **DDSDataReader** (p. 1556) is specified by the **DDS_ResourceLimitsQosPolicy** (p. 1308) and the **DDS_DataReaderResourceLimitsQosPolicy** (p. 867).

If the sequences are not empty (maximum size not equal to 0 and length not equal to 0) when **FooDataReader::take** (p. 1920) is called, samples are copied to received_data and info_seq. The application will not need to call **FooData↩ Reader::return_loan** (p. 1940).

The order of the samples returned to the caller depends on the **DDS_PresentationQosPolicy** (p. 1249).

- If **DDS_PresentationQosPolicy::access_scope** (p. 1252) is **DDS_INSTANCE_PRESENTATION_QOS** (p. 616), the returned collection is a list where samples belonging to the same data instance are consecutive.

- If **DDS_PresentationQosPolicy::access_scope** (p. 1252) is **DDS_TOPIC_PRESENTATION_QOS** (p. 616) and **DDS_PresentationQosPolicy::ordered_access** (p. 1252) is set to **DDS_BOOLEAN_FALSE** (p. 513), then the returned collection is a list where samples belonging to the same data instance are consecutive.

- If **DDS_PresentationQosPolicy::access_scope** (p. 1252) is **DDS_TOPIC_PRESENTATION_QOS** (p. 616) and **DDS_PresentationQosPolicy::ordered_access** (p. 1252) is set to **DDS_BOOLEAN_TRUE** (p. 513), then the returned collection is a list where the relative order of samples as they were written by a DataWriter is preserved also across different instances. In other words, changes made by a single DataWriter are made available to subscribers in the same order in which they occur. Samples belonging to the same instance may or may not be consecutive. This is because, to preserve order within a single DataWriter, it may be necessary to mix samples from different instances.

- If **DDS_PresentationQosPolicy::access_scope** (p. 1252) is **DDS_GROUP_PRESENTATION_QOS** (p. 616) and **DDS_PresentationQosPolicy::ordered_access** (p. 1252) is set to **DDS_BOOLEAN_FALSE** (p. 513), then the returned collection is a list where samples belonging to the same data instance are consecutive.

- If **DDS_PresentationQosPolicy::access_scope** (p. 1252) is **DDS_GROUP_PRESENTATION_QOS** (p. 616) and **DDS_PresentationQosPolicy::ordered_access** (p. 1252) is set to **DDS_BOOLEAN_TRUE** (p. 513), then changes made to instances by a set of DataWriters attached to a common Publisher are made available in the order in which they were written. For this to happen, the application must take the samples using the Subscriber's **DDSSubscriber::begin_access** (p. 1873) and **DDSSubscriber::end_access** (p. 1874) operations.

See the "Ordered Access" section in the "PRESENTATION QosPolicy" section of the Core Libraries User's Manual for more information.

In all of the above cases, the relative order between the samples of one instance is consistent with the **DESTINATION↩ _ORDER** (p. 584) policy:

- If **DDS_DestinationOrderQosPolicy::kind** (p. 933) is **DDS_BY_RECEPTION_TIMESTAMP_DESTINATIONORDER_QOS** (p. 585), samples belonging to the same instances will appear in the relative order in which there were received (FIFO, earlier samples ahead of the later samples).

- If **DDS_DestinationOrderQosPolicy::kind** (p. 933) is **DDS_BY_SOURCE_TIMESTAMP_DESTINATIONORDER_QOS** (p. 585), samples belonging to the same instances will appear in the relative order implied by the `source_timestamp` (FIFO, smaller values of `source_timestamp` ahead of the larger values).

If the **DDSDataReader** (p. 1556) has no samples that meet the constraints, the method will fail with **DDS_RETCODE_NO_DATA** (p. 533).

In addition to the collection of samples, the read and take operations also use a collection of **DDS_SampleInfo** (p. 1344) structures.

The initial (input) properties of the `received_data` and `info_seq` collections will determine the precise behavior of the read or take operation. For the purposes of this description, the collections are modeled as having these properties:

- whether the collection container owns the memory of the elements within (`owns`, see **FooSeq::has_ownership** (p. 1979))
- the current-length (`len`, see **FooSeq::length()** (p. 1973))
- the maximum length (`max_len`, see **FooSeq::maximum()** (p. 1975))

The initial values of the `owns`, `len` and `max_len` properties for the `received_data` and `info_seq` collections govern the behavior of the read and take operations as specified by the following rules:

1. The values of `owns`, `len` and `max_len` properties for the two collections must be identical. Otherwise read/take will fail with **DDS_RETCODE_PRECONDITION_NOT_MET** (p. 532).
2. On successful output, the values of `owns`, `len` and `max_len` will be the same for both collections.
3. If the initial `max_len==0`, then the `received_data` and `info_seq` collections will be filled with elements that are loaned by the **DDSDataReader** (p. 1556). On output, `owns` will be FALSE, `len` will be set to the number of values returned, and `max_len` will be set to a value verifying `max_len >= len`. The use of this variant allows for Zero Copy access to the data and the application will need to return the loan to the **DDSDataWriter** (p. 1589) using **FooDataReader::return_loan** (p. 1940).
4. If the initial `max_len>0` and `owns==FALSE`, then the read or take operation will fail with **DDS_RETCODE_PRECONDITION_NOT_MET** (p. 532). This avoids the potential hard-to-detect memory leaks caused by an application forgetting to return the loan.
5. If initial `max_len>0` and `owns==TRUE`, then the read or take operation will copy the `received_data` values and **DDS_SampleInfo** (p. 1344) values into the elements already inside the collections. On output, `owns` will be TRUE, `len` will be set to the number of values copied and `max_len` will remain unchanged. The use of this variant forces a copy but the application can control where the copy is placed and the application will not need to return the loan. The number of samples copied depends on the relative values of `max_len` and `max_samples`:
    - If `max_samples` == LENGTH_UNLIMITED, then at most `max_len` values will be copied. The use of this variant lets the application limit the number of samples returned to what the sequence can accommodate.

- If max_samples <= max_len, then at most max_samples values will be copied. The use of this variant lets the application limit the number of samples returned to fewer that what the sequence can accommodate.
- If max_samples > max_len, then the read or take operation will fail with **DDS_RETCODE_PRECONDITION_NOT_MET** (p. 532). This avoids the potential confusion where the application expects to be able to access up to max_samples, but that number can never be returned, even if they are available in the **DDSDataReader** (p. 1556), because the output sequence cannot accommodate them.

As described above, upon completion, the received_data and info_seq collections may contain elements loaned from the **DDSDataReader** (p. 1556). If this is the case, the application will need to use **FooDataReader::return_loan** (p. 1940) to return the loan once it is no longer using the received_data in the collection. When **FooDataReader::return_loan** (p. 1940) completes, the collection will have owns=FALSE and max_len=0. The application can determine whether it is necessary to return the loan or not based on how the state of the collections when the read/take operation was called or by accessing the owns property. However, in many cases it may be simpler to always call **FooDataReader::return_loan** (p. 1940), as this operation is harmless (i.e., it leaves all elements unchanged) if the collection does not have a loan. To avoid potential memory leaks, the implementation of the **Foo** (p. 1916) and **DDS_SampleInfo** (p. 1344) collections should disallow changing the length of a collection for which owns==FALSE. Furthermore, deleting a collection for which owns==FALSE should be considered an error.

On output, the collection of **Foo** (p. 1916) values and the collection of **DDS_SampleInfo** (p. 1344) structures are of the same length and are in a one-to-one correspondence. Each **DDS_SampleInfo** (p. 1344) provides information, such as the source_timestamp, the sample_state, view_state, and instance_state, etc., about the corresponding sample. Some elements in the returned collection may not have valid data. If the instance_state in the **DDS_SampleInfo** (p. 1344) is **DDS_NOT_ALIVE_DISPOSED_INSTANCE_STATE** (p. 166) or **DDS_NOT_ALIVE_NO_WRITERS_INSTANCE_STATE** (p. 166), then the last sample for that instance in the collection (that is, the one whose **DDS_SampleInfo** (p. 1344) has sample_rank==0) does not contain valid data.

Samples that contain no data do not count towards the limits imposed by the **DDS_ResourceLimitsQosPolicy** (p. 1308). The act of reading/taking a sample sets its sample_state to **DDS_READ_SAMPLE_STATE** (p. 162).

If the sample belongs to the most recent generation of the instance, it will also set the view_state of the instance to **DDS_NOT_NEW_VIEW_STATE** (p. 164). It will not affect the instance_state of the instance.

This operation must be provided on the specialized class that is generated for the particular application data-type that is being read (**Foo** (p. 1916)).

If the **DDSDataReader** (p. 1556) has no samples that meet the constraints, the operations fails with **DDS_RETCODE_NO_DATA** (p. 533). For an example on how take can be used, please refer to the **Access received data via a reader** (p. 404) "receive example".

Parameters

received_data	<<***inout***>> *(p. 433)* User data type-specific **FooSeq** (p. 1965) object where the received data samples will be returned.
info_seq	<<***inout***>> *(p. 433)* A **DDS_SampleInfoSeq** (p. 1355) object where the received sample info will be returned.
max_samples	<<***in***>> *(p. 433)* The maximum number of samples to be returned. If the special value **DDS_LENGTH_UNLIMITED** (p. 636) is provided, as many samples will be returned as are available, up to the limits described above.
sample_states	<<***in***>> *(p. 433)* Data samples matching one of these sample_states are returned.
view_states	<<***in***>> *(p. 433)* Data samples matching one of these view_state are returned.
instance_states	<<***in***>> *(p. 433)* Data samples matching one of these instance_state are returned.

Exceptions

One	of the **Standard Return Codes** (p. 532), **DDS_RETCODE_PRECONDITION_NOT_MET** (p. 532), **DDS_RETCODE_NO_DATA** (p. 533) or **DDS_RETCODE_NOT_ENABLED** (p. 533).

See also

FooDataReader::read (p. 1919)

FooDataReader::read_w_condition (p. 1925), **FooDataReader::take_w_condition** (p. 1926)

DDS_LENGTH_UNLIMITED (p. 636)

9.311.2.4 read_w_condition()

```
virtual  DDS_ReturnCode_t FooDataReader::read_w_condition (
              FooSeq & received_data,
              DDS_SampleInfoSeq & info_seq,
              DDS_Long max_samples,
              DDSReadCondition * condition )  [pure virtual]
```

Accesses via **FooDataReader::read** (p. 1919) the samples that match the criteria specified in the **DDSReadCondition** (p. 1842).

This operation is especially useful in combination with **DDSQueryCondition** (p. 1841) to filter data samples based on the content.

The specified **DDSReadCondition** (p. 1842) must be attached to the **DDSDataReader** (p. 1556); otherwise the operation will fail with **DDS_RETCODE_PRECONDITION_NOT_MET** (p. 532).

In case the **DDSReadCondition** (p. 1842) is a plain **DDSReadCondition** (p. 1842) and not the specialized **DDSQuery↩ Condition** (p. 1841), the operation is equivalent to calling **FooDataReader::read** (p. 1919) and passing as `sample↩ _states`, `view_states` and `instance_states` the value of the corresponding attributes in the `read_↩ condition`. Using this operation, the application can avoid repeating the same parameters specified when creating the **DDSReadCondition** (p. 1842).

The samples are accessed with the same semantics as **FooDataReader::read** (p. 1919).

If the **DDSDataReader** (p. 1556) has no samples that meet the constraints, the operation will fail with **DDS_RETCODE↩ _NO_DATA** (p. 533).

Parameters

received_data	*<<**inout**>> (p. 433)* user data type-specific **FooSeq** (p. 1965) object where the received data samples will be returned.
info_seq	*<<**inout**>> (p. 433)* a **DDS_SampleInfoSeq** (p. 1355) object where the received sample info will be returned.
max_samples	*<<**in**>> (p. 433)* The maximum number of samples to be returned. If the special value **DDS_LENGTH_UNLIMITED** (p. 636) is provided, as many samples will be returned as are available, up to the limits described in the documentation for **FooDataReader::take()** (p. 1920).
condition	*<<**in**>> (p. 433)* the **DDSReadCondition** (p. 1842) to select samples of interest. Cannot be NULL.

Exceptions

One	of the **Standard Return Codes** (p. 532), **DDS_RETCODE_PRECONDITION_NOT_MET** (p. 532), **DDS_RETCODE_NO_DATA** (p. 533) or **DDS_RETCODE_NOT_ENABLED** (p. 533).

See also

FooDataReader::read (p. 1919)

FooDataReader::take (p. 1920), **FooDataReader::take_w_condition** (p. 1926)

DDS_LENGTH_UNLIMITED (p. 636)

9.311.2.5 take_w_condition()

```
virtual  DDS_ReturnCode_t FooDataReader::take_w_condition (
              FooSeq & received_data,
              DDS_SampleInfoSeq & info_seq,
              DDS_Long max_samples,
              DDSReadCondition * condition )  [pure virtual]
```

Analogous to **FooDataReader::read_w_condition** (p. 1925) except it accesses samples via the **FooDataReader::take** (p. 1920) operation.

This operation is analogous to **FooDataReader::read_w_condition** (p. 1925) except that it accesses samples via the **FooDataReader::take** (p. 1920) operation.

The specified **DDSReadCondition** (p. 1842) must be attached to the **DDSDataReader** (p. 1556); otherwise the operation will fail with **DDS_RETCODE_PRECONDITION_NOT_MET** (p. 532).

The samples are accessed with the same semantics as **FooDataReader::take** (p. 1920).

This operation is especially useful in combination with **DDSQueryCondition** (p. 1841) to filter data samples based on the content.

If the **DDSDataReader** (p. 1556) has no samples that meet the constraints, the method will fail with **DDS_RETCODE_NO_DATA** (p. 533).

Parameters

received_data	*<<**inout**>> (p. 433)* user data type-specific **FooSeq** (p. 1965) object where the received data samples will be returned.
info_seq	*<<**inout**>> (p. 433)* a **DDS_SampleInfoSeq** (p. 1355) object where the received sample info will be returned.
max_samples	*<<**in**>> (p. 433)* The maximum number of samples to be returned. If the special value **DDS_LENGTH_UNLIMITED** (p. 636) is provided, as many samples will be returned as are available, up to the limits described in the documentation for **FooDataReader::take()** (p. 1920).
condition	*<<**in**>> (p. 433)* the **DDSReadCondition** (p. 1842) to select samples of interest. Cannot be NULL.

Exceptions

One	of the **Standard Return Codes** (p. 532), **DDS_RETCODE_PRECONDITION_NOT_MET** (p. 532), **DDS_RETCODE_NO_DATA** (p. 533) or **DDS_RETCODE_NOT_ENABLED** (p. 533).

See also

FooDataReader::read_w_condition (p. 1925), **FooDataReader::read** (p. 1919)

FooDataReader::take (p. 1920)

DDS_LENGTH_UNLIMITED (p. 636)

9.311.2.6 read_next_sample()

```
virtual  DDS_ReturnCode_t FooDataReader::read_next_sample (
              Foo & received_data,
              DDS_SampleInfo & sample_info )  [pure virtual]
```

Copies the next not-previously-accessed data value from the **DDSDataReader** (p. 1556).

This operation copies the next not-previously-accessed data value from the **DDSDataReader** (p. 1556). This operation also copies the corresponding **DDS_SampleInfo** (p. 1344). The implied order among the samples stored in the **DDSDataReader** (p. 1556) is the same as for the **FooDataReader::read** (p. 1919) operation.

The **FooDataReader::read_next_sample** (p. 1927) operation is semantically equivalent to the **FooDataReader::read** (p. 1919) operation, where the input data sequences has max_len=1, the sample_states=NOT_READ, the view_↩ states=ANY_VIEW_STATE, and the instance_states=ANY_INSTANCE_STATE.

The **FooDataReader::read_next_sample** (p. 1927) operation provides a simplified API to 'read' samples, avoiding the need for the application to manage sequences and specify states.

If there is no unread data in the **DDSDataReader** (p. 1556), the operation will fail with **DDS_RETCODE_NO_DATA** (p. 533) and nothing is copied.

Note

Calling **FooDataReader::read_next_sample** (p. 1927) from the **DDSDataReaderListener::on_data_available()** (p. 1585) callback reads only one sample of potentially many samples in the reader queue, because **DDSData↩ ReaderListener::on_data_available()** (p. 1585) is triggered only once when new samples arrive in the queue. Therefore, it is recommended that you call **FooDataReader::read_next_sample** (p. 1927) in a loop within the on_data_available callback until **FooDataReader::read_next_sample** (p. 1927) returns **DDS_RETCODE_NO↩ _DATA** (p. 533) . This ensures that all samples in the reader queue are serviced by application logic. (You may also choose to use **FooDataReader::read** (p. 1919) rather than **FooDataReader::read_next_sample** (p. 1927) in order to read more than one sample at a time.)

Parameters

received_data	*<<**inout**>> (p. 433)* user data type-specific **Foo** (p. 1916) object where the next received data sample will be returned. The received_data must have been fully allocated. Otherwise, this operation may fail.
sample_info	*<<**inout**>> (p. 433)* a **DDS_SampleInfo** (p. 1344) object where the next received sample info will be returned.

Exceptions

One	of the **Standard Return Codes** (p. 532), **DDS_RETCODE_NO_DATA** (p. 533) or **DDS_RETCODE_NOT_ENABLED** (p. 533).

See also

FooDataReader::read (p. 1919)

9.311.2.7 take_next_sample()

```
virtual  DDS_ReturnCode_t FooDataReader::take_next_sample (
              Foo & received_data,
              DDS_SampleInfo & sample_info )  [pure virtual]
```

Copies the next not-previously-accessed data value from the **DDSDataReader** (p. 1556).

This operation copies the next not-previously-accessed data value from the **DDSDataReader** (p. 1556) and 'removes' it from the **DDSDataReader** (p. 1556) so that it is no longer accessible. This operation also copies the corresponding **DDS_SampleInfo** (p. 1344). This operation is analogous to the **FooDataReader::read_next_sample** (p. 1927) except for the fact that the sample is removed from the **DDSDataReader** (p. 1556).

The **FooDataReader::take_next_sample** (p. 1928) operation is semantically equivalent to the **FooDataReader::take** (p. 1920) operation, where the input data sequences has max_len=1, the sample_states=NOT_READ, the view_↩ states=ANY_VIEW_STATE, and the instance_states=ANY_INSTANCE_STATE.

The **FooDataReader::take_next_sample** (p. 1928) operation provides a simplified API to 'take' samples, avoiding the need for the application to manage sequences and specify states.

If there is no unread data in the **DDSDataReader** (p. 1556), the operation will fail with **DDS_RETCODE_NO_DATA** (p. 533) and nothing is copied.

Note

Calling **FooDataReader::take_next_sample** (p. 1928) from the **DDSDataReaderListener::on_data_available()** (p. 1585) callback retrieves only one sample of potentially many samples in the reader queue, because **DDSData↩ ReaderListener::on_data_available()** (p. 1585) is triggered only once when new samples arrive in the queue. Therefore, it is recommended that you call **FooDataReader::take_next_sample** (p. 1928) in a loop within the on_data_available callback until **FooDataReader::take_next_sample** (p. 1928) returns **DDS_RETCODE_NO_↩ DATA** (p. 533) . This ensures that all samples in the reader queue are serviced by application logic. (You may also choose to use the **FooDataReader::take** (p. 1920) rather than **FooDataReader::take_next_sample** (p. 1928) in order to take more than one sample at a time.)

Parameters

received_data	*<<**inout**>> (p. 433)* user data type-specific **Foo** (p. 1916) object where the next received data sample will be returned. The received_data must have been fully allocated. Otherwise, this operation may fail.
sample_info	*<<**inout**>> (p. 433)* a **DDS_SampleInfo** (p. 1344) object where the next received sample info will be returned.

Exceptions

One	of the **Standard Return Codes** (p. 532),

DDS_RETCODE_NO_DATA (p. 533) or **DDS_RETCODE_NOT_ENABLED** (p. 533).

See also

FooDataReader::take (p. 1920)

9.311.2.8 read_instance()

```
virtual  DDS_ReturnCode_t FooDataReader::read_instance (
              FooSeq & received_data,
              DDS_SampleInfoSeq & info_seq,
              DDS_Long max_samples,
             const  DDS_InstanceHandle_t & a_handle,
              DDS_SampleStateMask sample_states,
              DDS_ViewStateMask view_states,
              DDS_InstanceStateMask instance_states )  [pure virtual]
```

Access a collection of data samples from the **DDSDataReader** (p. 1556).

This operation accesses a collection of data values from the **DDSDataReader** (p. 1556). The behavior is identical to **FooDataReader::read** (p. 1919), except that all samples returned belong to the single specified instance whose handle is `a_handle`.

Upon successful completion, the data collection will contain samples all belonging to the same instance. The corresponding **DDS_SampleInfo** (p. 1344) verifies **DDS_SampleInfo::instance_handle** (p. 1349) == `a_handle`.

The **FooDataReader::read_instance** (p. 1929) operation is semantically equivalent to the **FooDataReader::read** (p. 1919) operation, except in building the collection, the **DDSDataReader** (p. 1556) will check that the sample belongs to the specified instance and otherwise it will not place the sample in the returned collection.

The behavior of the **FooDataReader::read_instance** (p. 1929) operation follows the same rules as the **FooData↩ Reader::read** (p. 1919) operation regarding the pre-conditions and post-conditions for the `received_data` and `sample_info`. Similar to the **FooDataReader::read** (p. 1919), the **FooDataReader::read_instance** (p. 1929) operation may 'loan' elements to the output collections, which must then be returned by means of **FooDataReader↩ ::return_loan** (p. 1940).

Similar to the **FooDataReader::read** (p. 1919), this operation must be provided on the specialized class that is generated for the particular application data-type that is being taken.

If the **DDSDataReader** (p. 1556) has no samples that meet the constraints, the method will fail with **DDS_RETCODE_NO_DATA** (p. 533).

This operation may fail with **DDS_RETCODE_BAD_PARAMETER** (p. 532) if the **DDS_InstanceHandle_t** (p. 76) `a_handle` does not correspond to an existing data-object known to the **DDSDataReader** (p. 1556).

Parameters

received_data	*<<**inout**>> (p. 433)* user data type-specific **FooSeq** (p. 1965) object where the received data samples will be returned.
info_seq	*<<**inout**>> (p. 433)* a **DDS_SampleInfoSeq** (p. 1355) object where the received sample info will be returned.
max_samples	*<<**in**>> (p. 433)* The maximum number of samples to be returned. If the special value **DDS_LENGTH_UNLIMITED** (p. 636) is provided, as many samples will be returned as are available, up to the limits described in the documentation for **FooDataReader::take()** (p. 1920).
a_handle	*<<**in**>> (p. 433)* The specified instance to return samples for. The method will fail with **DDS_RETCODE_BAD_PARAMETER** (p. 532) if the `handle` does not correspond to an existing data-object known to the **DDSDataReader** (p. 1556).
sample_states	*<<**in**>> (p. 433)* data samples matching ones of these `sample_states` are returned.
view_states	*<<**in**>> (p. 433)* data samples matching ones of these `view_state` are returned.
instance_states	*<<**in**>> (p. 433)* data samples matching ones of these `instance_state` are returned.

Exceptions

One	of the **Standard Return Codes** (p. 532), **DDS_RETCODE_PRECONDITION_NOT_MET** (p. 532), **DDS_RETCODE_NO_DATA** (p. 533) or **DDS_RETCODE_NOT_ENABLED** (p. 533).

See also

FooDataReader::read (p. 1919)

DDS_LENGTH_UNLIMITED (p. 636)

9.311.2.9 take_instance()

```
virtual  DDS_ReturnCode_t FooDataReader::take_instance (
              FooSeq & received_data,
              DDS_SampleInfoSeq & info_seq,
              DDS_Long max_samples,
             const  DDS_InstanceHandle_t & a_handle,
              DDS_SampleStateMask sample_states,
              DDS_ViewStateMask view_states,
              DDS_InstanceStateMask instance_states )  [pure virtual]
```

Access a collection of data samples from the **DDSDataReader** (p. 1556).

This operation accesses a collection of data values from the **DDSDataReader** (p. 1556). The behavior is identical to **FooDataReader::take** (p. 1920), except for that all samples returned belong to the single specified instance whose handle is `a_handle`.

The semantics are the same for the **FooDataReader::take** (p. 1920) operation, except in building the collection, the **DDSDataReader** (p. 1556) will check that the sample belongs to the specified instance, and otherwise it will not place the sample in the returned collection.

The behavior of the **FooDataReader::take_instance** (p. 1931) operation follows the same rules as the **FooData↩ Reader::read** (p. 1919) operation regarding the pre-conditions and post-conditions for the `received_data` and `sample_info`. Similar to the **FooDataReader::read** (p. 1919), the **FooDataReader::take_instance** (p. 1931) operation may 'loan' elements to the output collections, which must then be returned by means of **FooDataReader↩ ::return_loan** (p. 1940).

Similar to the **FooDataReader::read** (p. 1919), this operation must be provided on the specialized class that is generated for the particular application data-type that is being taken.

If the **DDSDataReader** (p. 1556) has no samples that meet the constraints, the method fails with **DDS_RETCODE_↩ NO_DATA** (p. 533).

This operation may fail with **DDS_RETCODE_BAD_PARAMETER** (p. 532) if the **DDS_InstanceHandle_t** (p. 76) `a_↩ handle` does not correspond to an existing data-object known to the **DDSDataReader** (p. 1556).

Parameters

received_data	*<<**inout**>> (p. 433)* user data type-specific **FooSeq** (p. 1965) object where the received data samples will be returned.
info_seq	*<<**inout**>> (p. 433)* a **DDS_SampleInfoSeq** (p. 1355) object where the received sample info will be returned.
max_samples	*<<**in**>> (p. 433)* The maximum number of samples to be returned. If the special value **DDS_LENGTH_UNLIMITED** (p. 636) is provided, as many samples will be returned as are available, up to the limits described in the documentation for **FooDataReader::take()** (p. 1920).
a_handle	*<<**in**>> (p. 433)* The specified instance to return samples for. The method will fail with **DDS_RETCODE_BAD_PARAMETER** (p. 532) if the `handle` does not correspond to an existing data-object known to the **DDSDataReader** (p. 1556).
sample_states	*<<**in**>> (p. 433)* data samples matching ones of these `sample_states` are returned.
view_states	*<<**in**>> (p. 433)* data samples matching ones of these `view_state` are returned.
instance_states	*<<**in**>> (p. 433)* data samples matching ones of these `instance_state` are returned.

Exceptions

One	of the **Standard Return Codes** (p. 532), **DDS_RETCODE_PRECONDITION_NOT_MET** (p. 532), **DDS_RETCODE_NO_DATA** (p. 533) or **DDS_RETCODE_NOT_ENABLED** (p. 533).

See also

FooDataReader::take (p. 1920)

DDS_LENGTH_UNLIMITED (p. 636)

9.311.2.10 read_instance_w_condition()

```
virtual DDS_ReturnCode_t FooDataReader::read_instance_w_condition (
            FooSeq & received_data,
```

```
DDS_SampleInfoSeq & info_seq,
DDS_Long max_samples,
const DDS_InstanceHandle_t & previous_handle,
DDSReadCondition * condition ) [pure virtual]
```

<<***extension***>> *(p. 432)* Accesses via **FooDataReader::read_instance** (p. 1929) the samples that match the criteria specified in the **DDSReadCondition** (p. 1842).

This operation accesses a collection of data values from the **DDSDataReader** (p. 1556). The behavior is identical to **FooDataReader::read_instance** (p. 1929), except that all returned samples satisfy the specified condition. In other words, on success, all returned samples belong to belong the single specified instance whose handle is `a_handle`, and for which the specified **DDSReadCondition** (p. 1842) evaluates to TRUE.

The behavior of the **FooDataReader::read_instance_w_condition** (p. 1932) operation follows the same rules as the **FooDataReader::read** (p. 1919) operation regarding the pre-conditions and post-conditions for the `received_data` and `sample_info`. Similar to the **FooDataReader::read** (p. 1919), the **FooDataReader::read_instance_w_condition** (p. 1932) operation may 'loan' elements to the output collections, which must then be returned by means of **FooDataReader::return_loan** (p. 1940).

Similar to **FooDataReader::read** (p. 1919), this operation must be provided on the specialized class that is generated for the particular application data-type that is being taken.

If the **DDSDataReader** (p. 1556) has no samples that meet the constraints, the method will fail with **DDS_RETCODE_NO_DATA** (p. 533).

Parameters

received_data	<<***inout***>> *(p. 433)* user data type-specific **FooSeq** (p. 1965) object where the received data samples will be returned.
info_seq	<<***inout***>> *(p. 433)* a **DDS_SampleInfoSeq** (p. 1355) object where the received sample info will be returned.
max_samples	<<***in***>> *(p. 433)* The maximum number of samples to be returned. If the special value **DDS_LENGTH_UNLIMITED** (p. 636) is provided, as many samples will be returned as are available, up to the limits described in the documentation for **FooDataReader::take()** (p. 1920).
previous_handle	<<***in***>> *(p. 433)* The 'next smallest' instance with a value greater than this value that has available samples will be returned.
condition	<<***in***>> *(p. 433)* the **DDSReadCondition** (p. 1842) to select samples of interest. Cannot be NULL.

Exceptions

One	of the **Standard Return Codes** (p. 532), **DDS_RETCODE_PRECONDITION_NOT_MET** (p. 532) **DDS_RETCODE_NO_DATA** (p. 533) or **DDS_RETCODE_NOT_ENABLED** (p. 533).

See also

FooDataReader::read_next_instance (p. 1935)

DDS_LENGTH_UNLIMITED (p. 636)

9.311.2.11 take_instance_w_condition()

```
virtual DDS_ReturnCode_t FooDataReader::take_instance_w_condition (
              FooSeq & received_data,
              DDS_SampleInfoSeq & info_seq,
              DDS_Long max_samples,
             const DDS_InstanceHandle_t & previous_handle,
              DDSReadCondition * condition ) [pure virtual]
```

<<***extension***>> *(*p. *432)* Accesses via **FooDataReader::take_instance** (p. 1931) the samples that match the criteria specified in the **DDSReadCondition** (p. 1842).

This operation accesses a collection of data values from the **DDSDataReader** (p. 1556) and 'removes' them from the **DDSDataReader** (p. 1556). The behavior is identical to **FooDataReader::take_instance** (p. 1931), except that all returned samples satisfy the specified condition. In other words, on success, all returned samples belong to belong the single specified instance whose handle is `a_handle`, and for which the specified **DDSReadCondition** (p. 1842) evaluates to TRUE.

The operation has the same behavior as **FooDataReader::read_instance_w_condition** (p. 1932), except that the samples are 'taken' from the **DDSDataReader** (p. 1556) such that they are no longer accessible via subsequent 'read' or 'take' operations.

The behavior of the **FooDataReader::take_instance_w_condition** (p. 1933) operation follows the same rules as the **FooDataReader::read** (p. 1919) operation regarding the pre-conditions and post-conditions for the `received_data` and `sample_info`. Similar to the **FooDataReader::read** (p. 1919), the **FooDataReader::take_instance_w_condition** (p. 1933) operation may 'loan' elements to the output collections, which must then be returned by means of **FooDataReader::return_loan** (p. 1940).

Similar to the **FooDataReader::read** (p. 1919), this operation must be provided on the specialized class that is generated for the particular application data-type that is being taken.

If the **DDSDataReader** (p. 1556) has no samples that meet the constraints, the method will fail with **DDS_RETCODE_NO_DATA** (p. 533).

Parameters

received_data	<<***inout***>> *(*p. *433)* user data type-specific **FooSeq** (p. 1965) object where the received data samples will be returned.
info_seq	<<***inout***>> *(*p. *433)* a **DDS_SampleInfoSeq** (p. 1355) object where the received sample info will be returned.
max_samples	<<***in***>> *(*p. *433)* The maximum number of samples to be returned. If the special value **DDS_LENGTH_UNLIMITED** (p. 636) is provided, as many samples will be returned as are available, up to the limits described in the documentation for **FooDataReader::take()** (p. 1920).
previous_handle	<<***in***>> *(*p. *433)* The 'next smallest' instance with a value greater than this value that has available samples will be returned.
condition	<<***in***>> *(*p. *433)* the **DDSReadCondition** (p. 1842) to select samples of interest. Cannot be NULL.

Exceptions

One	of the **Standard Return Codes** (p. 532), **DDS_RETCODE_PRECONDITION_NOT_MET** (p. 532), or **DDS_RETCODE_NO_DATA** (p. 533), **DDS_RETCODE_NOT_ENABLED** (p. 533).

See also

FooDataReader::take_next_instance (p. 1936)

DDS_LENGTH_UNLIMITED (p. 636)

9.311.2.12 read_next_instance()

```
virtual  DDS_ReturnCode_t FooDataReader::read_next_instance (
              FooSeq & received_data,
              DDS_SampleInfoSeq & info_seq,
              DDS_Long max_samples,
             const  DDS_InstanceHandle_t & previous_handle,
              DDS_SampleStateMask sample_states,
              DDS_ViewStateMask view_states,
              DDS_InstanceStateMask instance_states )  [pure virtual]
```

Access a collection of data samples from the **DDSDataReader** (p. 1556).

This operation accesses a collection of data values from the **DDSDataReader** (p. 1556) where all the samples belong to a single instance. The behavior is similar to **FooDataReader::read_instance** (p. 1929), except that the actual instance is not directly specified. Rather, the samples will all belong to the 'next' instance with `instance_handle` 'greater' than the specified 'previous_handle' that has available samples.

This operation implies the existence of a total order 'greater-than' relationship between the instance handles. The specifics of this relationship are not all important and are implementation specific. The important thing is that, according to the middleware, all instances are ordered relative to each other. This ordering is between the instance handles; It should not depend on the state of the instance (e.g. whether it has data or not) and must be defined even for instance handles that do not correspond to instances currently managed by the **DDSDataReader** (p. 1556). For the purposes of the ordering, it should be 'as if' each instance handle was represented as unique integer.

The behavior of **FooDataReader::read_next_instance** (p. 1935) is 'as if' the **DDSDataReader** (p. 1556) invoked **FooDataReader::read_instance** (p. 1929), passing the smallest instance_handle among all the ones that: (a) are greater than `previous_handle`, and (b) have available samples (i.e. samples that meet the constraints imposed by the specified states).

The special value **DDS_HANDLE_NIL** (p. 78) is guaranteed to be 'less than' any valid `instance_handle`. So the use of the parameter value `previous_handle` == **DDS_HANDLE_NIL** (p. 78) will return the samples for the instance which has the smallest `instance_handle` among all the instances that contain available samples.

Note

The operation **FooDataReader::read_next_instance** (p. 1935) is intended to be used in an application-driven iteration, where the application starts by passing `previous_handle` == **DDS_HANDLE_NIL** (p. 78), examines the samples returned, and then uses the `instance_handle` returned in the **DDS_SampleInfo** (p. 1344) as the value of the `previous_handle` argument to the next call to **FooDataReader::read_next_instance** (p. 1935). The iteration continues until **FooDataReader::read_next_instance** (p. 1935) fails with **DDS_RETCODE_NO_DATA** (p. 533). This application-driven iteration is required to ensure that all samples on the reader queue are read.

Note that it is possible to call the **FooDataReader::read_next_instance** (p. 1935) operation with a `previous_handle` that does not correspond to an instance currently managed by the **DDSDataReader** (p. 1556). This is because as stated earlier the 'greater-than' relationship is defined even for handles not managed by the **DDSDataReader** (p. 1556). One practical situation where this may occur is when an application is iterating though all the instances, takes all the samples of a **DDS_NOT_ALIVE_NO_WRITERS_INSTANCE_STATE** (p. 166) instance, returns the loan (at which point the instance information may be removed, and thus the handle becomes invalid), and tries to read the next instance. The behavior of the **FooDataReader::read_next_instance** (p. 1935) operation follows the same rules as the **FooDataReader::read** (p. 1919) operation regarding the pre-conditions and post-conditions for the `received_data` and `sample_info`. Similar to the **FooDataReader::read** (p. 1919), the **FooDataReader::read_instance** (p. 1929) operation may 'loan' elements to the output collections, which must then be returned by means of **FooDataReader::return_loan** (p. 1940).

Similar to the **FooDataReader::read** (p. 1919), this operation must be provided on the specialized class that is generated for the particular application data-type that is being taken.

If the **DDSDataReader** (p. 1556) has no samples that meet the constraints, the method will fail with **DDS_RETCODE_NO_DATA** (p. 533).

Parameters

received_data	<<***inout***>> *(p. 433)* user data type-specific **FooSeq** (p. 1965) object where the received data samples will be returned.
info_seq	<<***inout***>> *(p. 433)* a **DDS_SampleInfoSeq** (p. 1355) object where the received sample info will be returned.
max_samples	<<***in***>> *(p. 433)* The maximum number of samples to be returned. If the special value **DDS_LENGTH_UNLIMITED** (p. 636) is provided, as many samples will be returned as are available, up to the limits described in the documentation for **FooDataReader::take()** (p. 1920).
previous_handle	<<***in***>> *(p. 433)* The 'next smallest' instance with a value greater than this value that has available samples will be returned.
sample_states	<<***in***>> *(p. 433)* data samples matching ones of these `sample_states` are returned.
view_states	<<***in***>> *(p. 433)* data samples matching ones of these `view_state` are returned.
instance_states	<<***in***>> *(p. 433)* data samples matching ones of these `instance_state` are returned.

Exceptions

One	of the **Standard Return Codes** (p. 532), **DDS_RETCODE_PRECONDITION_NOT_MET** (p. 532) **DDS_RETCODE_NO_DATA** (p. 533) or **DDS_RETCODE_NOT_ENABLED** (p. 533).

See also

FooDataReader::read (p. 1919)

DDS_LENGTH_UNLIMITED (p. 636)

9.311.2.13 take_next_instance()

```
virtual DDS_ReturnCode_t FooDataReader::take_next_instance (
            FooSeq & received_data,
```

```
DDS_SampleInfoSeq & info_seq,
DDS_Long max_samples,
const  DDS_InstanceHandle_t & previous_handle,
DDS_SampleStateMask sample_states,
DDS_ViewStateMask view_states,
DDS_InstanceStateMask instance_states )  [pure virtual]
```

Access a collection of data samples from the **DDSDataReader** (p. 1556).

This operation accesses a collection of data values from the **DDSDataReader** (p. 1556) and 'removes' them from the **DDSDataReader** (p. 1556).

This operation has the same behavior as **FooDataReader::read_next_instance** (p. 1935), except that the samples are 'taken' from the **DDSDataReader** (p. 1556) such that they are no longer accessible via subsequent 'read' or 'take' operations.

Note

Like **FooDataReader::read_next_instance** (p. 1935), this operation is intended to be used in an application-driven iteration until all samples on the reader queue are taken. The iteration continues until **FooDataReader::take_next_instance** (p. 1936) fails with the value **DDS_RETCODE_NO_DATA** (p. 533) .

Similar to the operation **FooDataReader::read_next_instance** (p. 1935), it is possible to call **FooDataReader::take_next_instance** (p. 1936) with a `previous_handle` that does not correspond to an instance currently managed by the **DDSDataReader** (p. 1556).

The behavior of the **FooDataReader::take_next_instance** (p. 1936) operation follows the same rules as the **FooDataReader::read** (p. 1919) operation regarding the pre-conditions and post-conditions for the `received_data` and `sample_info`. Similar to the **FooDataReader::read** (p. 1919), the **FooDataReader::take_next_instance** (p. 1936) operation may 'loan' elements to the output collections, which must then be returned by means of **FooDataReader::return_loan** (p. 1940).

Similar to the **FooDataReader::read** (p. 1919), this operation must be provided on the specialized class that is generated for the particular application data-type that is being taken.

If the **DDSDataReader** (p. 1556) has no samples that meet the constraints, the method will fail with **DDS_RETCODE_NO_DATA** (p. 533).

Parameters

received_data	*<<**inout**>> (p. 433)* user data type-specific **FooSeq** (p. 1965) object where the received data samples will be returned.
info_seq	*<<**inout**>> (p. 433)* a **DDS_SampleInfoSeq** (p. 1355) object where the received sample info will be returned.
max_samples	*<<**in**>> (p. 433)* The maximum number of samples to be returned. If the special value **DDS_LENGTH_UNLIMITED** (p. 636) is provided, as many samples will be returned as are available, up to the limits described in the documentation for **FooDataReader::take()** (p. 1920).
previous_handle	*<<**in**>> (p. 433)* The 'next smallest' instance with a value greater than this value that has available samples will be returned.
sample_states	*<<**in**>> (p. 433)* data samples matching ones of these `sample_states` are returned.
view_states	*<<**in**>> (p. 433)* data samples matching ones of these `view_state` are returned.
instance_states	*<<**in**>> (p. 433)* data samples matching ones of these `instance_state` are returned.

Exceptions

One	of the **Standard Return Codes** (p. 532), **DDS_RETCODE_PRECONDITION_NOT_MET** (p. 532), **DDS_RETCODE_NO_DATA** (p. 533) or **DDS_RETCODE_NOT_ENABLED** (p. 533).

See also

FooDataReader::take (p. 1920)

DDS_LENGTH_UNLIMITED (p. 636)

9.311.2.14 read_next_instance_w_condition()

```
virtual  DDS_ReturnCode_t FooDataReader::read_next_instance_w_condition (
              FooSeq & received_data,
              DDS_SampleInfoSeq & info_seq,
              DDS_Long max_samples,
             const  DDS_InstanceHandle_t & previous_handle,
              DDSReadCondition * condition )  [pure virtual]
```

Accesses via **FooDataReader::read_next_instance** (p. 1935) the samples that match the criteria specified in the **DDSReadCondition** (p. 1842).

This operation accesses a collection of data values from the **DDSDataReader** (p. 1556). The behavior is identical to **FooDataReader::read_next_instance** (p. 1935), except that all returned samples satisfy the specified condition. In other words, on success, all returned samples belong to the same instance, and the instance is the instance with 'smallest' `instance_handle` among the ones that verify: (a) `instance_handle` >= `previous_handle`, and (b) have samples for which the specified **DDSReadCondition** (p. 1842) evaluates to TRUE.

Similar to the operation **FooDataReader::read_next_instance** (p. 1935), it is possible to call **FooDataReader::read_next_instance_w_condition** (p. 1938) with a `previous_handle` that does not correspond to an instance currently managed by the **DDSDataReader** (p. 1556).

The behavior of the **FooDataReader::read_next_instance_w_condition** (p. 1938) operation follows the same rules as the **FooDataReader::read** (p. 1919) operation regarding the pre-conditions and post-conditions for the `received_data` and `sample_info`. Similar to the **FooDataReader::read** (p. 1919), the **FooDataReader::read_next_instance_w_condition** (p. 1938) operation may 'loan' elements to the output collections, which must then be returned by means of **FooDataReader::return_loan** (p. 1940).

Similar to the **FooDataReader::read** (p. 1919), this operation must be provided on the specialized class that is generated for the particular application data-type that is being taken.

If the **DDSDataReader** (p. 1556) has no samples that meet the constraints, the method will fail with **DDS_RETCODE_NO_DATA** (p. 533).

Parameters

received_data	<<***inout***>> *(p. 433)* user data type-specific **FooSeq** (p. 1965) object where the received data samples will be returned.

Parameters

info_seq	*<<**inout**>> (p. 433)* a **DDS_SampleInfoSeq** (p. 1355) object where the received sample info will be returned.
max_samples	*<<**in**>> (p. 433)* The maximum number of samples to be returned. If the special value **DDS_LENGTH_UNLIMITED** (p. 636) is provided, as many samples will be returned as are available, up to the limits described in the documentation for **FooDataReader::take()** (p. 1920).
previous_handle	*<<**in**>> (p. 433)* The 'next smallest' instance with a value greater than this value that has available samples will be returned.
condition	*<<**in**>> (p. 433)* the **DDSReadCondition** (p. 1842) to select samples of interest. Cannot be NULL.

Exceptions

One	of the **Standard Return Codes** (p. 532), **DDS_RETCODE_PRECONDITION_NOT_MET** (p. 532) **DDS_RETCODE_NO_DATA** (p. 533) or **DDS_RETCODE_NOT_ENABLED** (p. 533).

See also

FooDataReader::read_next_instance (p. 1935)

DDS_LENGTH_UNLIMITED (p. 636)

9.311.2.15 take_next_instance_w_condition()

```
virtual  DDS_ReturnCode_t FooDataReader::take_next_instance_w_condition (
              FooSeq & received_data,
              DDS_SampleInfoSeq & info_seq,
              DDS_Long max_samples,
             const  DDS_InstanceHandle_t & previous_handle,
              DDSReadCondition * condition )  [pure virtual]
```

Accesses via **FooDataReader::take_next_instance** (p. 1936) the samples that match the criteria specified in the **DDSReadCondition** (p. 1842).

This operation accesses a collection of data values from the **DDSDataReader** (p. 1556) and 'removes' them from the **DDSDataReader** (p. 1556).

The operation has the same behavior as **FooDataReader::read_next_instance_w_condition** (p. 1938), except that the samples are 'taken' from the **DDSDataReader** (p. 1556) such that they are no longer accessible via subsequent 'read' or 'take' operations.

Similar to the operation **FooDataReader::read_next_instance** (p. 1935), it is possible to call **FooDataReader::take_next_instance_w_condition** (p. 1939) with a `previous_handle` that does not correspond to an instance currently managed by the **DDSDataReader** (p. 1556).

The behavior of the **FooDataReader::take_next_instance_w_condition** (p. 1939) operation follows the same rules as the **FooDataReader::read** (p. 1919) operation regarding the pre-conditions and post-conditions for the `received_data` and `sample_info`. Similar to **FooDataReader::read** (p. 1919), the **FooDataReader::take_next_instance_w_condition** (p. 1939) operation may 'loan' elements to the output collections, which must then be returned by means of **FooDataReader::return_loan** (p. 1940).

Similar to the **FooDataReader::read** (p. 1919), this operation must be provided on the specialized class that is generated for the particular application data-type that is being taken.

If the **DDSDataReader** (p. 1556) has no samples that meet the constraints, the method will fail with **DDS_RETCODE_NO_DATA** (p. 533).

Parameters

received_data	*<<**inout**>> (p. 433)* user data type-specific **FooSeq** (p. 1965) object where the received data samples will be returned.
info_seq	*<<**inout**>> (p. 433)* a **DDS_SampleInfoSeq** (p. 1355) object where the received sample info will be returned.
max_samples	*<<**in**>> (p. 433)* The maximum number of samples to be returned. If the special value **DDS_LENGTH_UNLIMITED** (p. 636) is provided, as many samples will be returned as are available, up to the limits described in the documentation for **FooDataReader::take()** (p. 1920).
previous_handle	*<<**in**>> (p. 433)* The 'next smallest' instance with a value greater than this value that has available samples will be returned.
condition	*<<**in**>> (p. 433)* the **DDSReadCondition** (p. 1842) to select samples of interest. Cannot be NULL.

Exceptions

One	of the **Standard Return Codes** (p. 532), **DDS_RETCODE_PRECONDITION_NOT_MET** (p. 532), or **DDS_RETCODE_NO_DATA** (p. 533), **DDS_RETCODE_NOT_ENABLED** (p. 533).

See also

FooDataReader::take_next_instance (p. 1936)

DDS_LENGTH_UNLIMITED (p. 636)

9.311.2.16 return_loan()

```
virtual  DDS_ReturnCode_t FooDataReader::return_loan (
              FooSeq & received_data,
              DDS_SampleInfoSeq & info_seq )  [pure virtual]
```

Indicates to the **DDSDataReader** (p. 1556) that the application is done accessing the collection of `received_data` and `info_seq` obtained by some earlier invocation of read or take on the **DDSDataReader** (p. 1556).

The `received_data` and `info_seq` must belong to a single related "pair"; that is, they should correspond to a pair returned from a single call to read or take. The `received_data` and `info_seq` must also have been obtained from

the same **DDSDataReader** (p. 1556) to which they are returned. If either of these conditions is not met, the operation will fail with **DDS_RETCODE_PRECONDITION_NOT_MET** (p. 532).

The operation **FooDataReader::return_loan** (p. 1940) allows implementations of the read and take operations to "loan" buffers from the **DDSDataReader** (p. 1556) to the application and in this manner provide "zerocopy" access to the data. During the loan, the **DDSDataReader** (p. 1556) will guarantee that the data and sample-information are not modified.

It is not necessary for an application to return the loans immediately after the read or take calls. However, as these buffers correspond to internal resources inside the **DDSDataReader** (p. 1556), the application should not retain them indefinitely.

The use of **FooDataReader::return_loan** (p. 1940) is only necessary if the read or take calls "loaned" buffers to the application. This only occurs if the `received_data` and `info_Seq` collections had `max_len=0` at the time read or take was called.

The application may also examine the "owns" property of the collection to determine where there is an outstanding loan. However, calling **FooDataReader::return_loan** (p. 1940) on a collection that does not have a loan is safe and has no side effects.

If the collections had a loan, upon completion of **FooDataReader::return_loan** (p. 1940), the collections will have `max_len=0`.

Similar to read, this operation must be provided on the specialized class that is generated for the particular application data-type that is being taken.

Parameters

received_data	*<<**in**>> (p. 433)* user data type-specific **FooSeq** (p. 1965) object where the received data samples was obtained from earlier invocation of read or take on the **DDSDataReader** (p. 1556).

Parameters

info_seq	*<<**in**>> (p. 433)* a **DDS_SampleInfoSeq** (p. 1355) object where the received sample info was obtained from earlier invocation of read or take on the **DDSDataReader** (p. 1556).

Exceptions

One	of the **Standard Return Codes** (p. 532), **DDS_RETCODE_PRECONDITION_NOT_MET** (p. 532) or **DDS_RETCODE_NOT_ENABLED** (p. 533).

9.311.2.17 get_key_value()

```
virtual  DDS_ReturnCode_t FooDataReader::get_key_value (
              Foo & key_holder,
            const  DDS_InstanceHandle_t handle )  [pure virtual]
```

Retrieve the instance `key` that corresponds to an instance `handle`.

Useful for keyed data types.

The operation will only fill the fields that form the `key` inside the `key_holder` instance. If the type has no keys, this method has no effect and exits with no error.

For keyed data types, this operation may fail with **DDS_RETCODE_BAD_PARAMETER** (p. 532) if the `handle` does not correspond to an existing data-object known to the **DDSDataReader** (p. 1556).

Parameters

key_holder	<<***inout***>> *(p. 433)* a user data type specific key holder, whose `key` fields are filled by this operation. If **Foo** (p. 1916) has no key, this method has no effect.
handle	<<***in***>> *(p. 433)* the `instance` whose key is to be retrieved. If **Foo** (p. 1916) *has* a key, `handle` must represent an existing instance of type **Foo** (p. 1916) known to the **DDSDataReader** (p. 1556). Otherwise, this method will fail with **DDS_RETCODE_BAD_PARAMETER** (p. 532).

If **Foo** (p. 1916) *has* a key and `handle` is **DDS_HANDLE_NIL** (p. 78), this method will fail with **DDS_RETCODE_BAD_PARAMETER** (p. 532).

If **Foo** (p. 1916) has a key and `handle` represents an instance of another type or an instance of type **Foo** (p. 1916) that has been unregistered, this method will fail with **DDS_RETCODE_BAD_PARAMETER** (p. 532). If **Foo** (p. 1916) has no key, this method has no effect.

Exceptions

One	of the **Standard Return Codes** (p. 532) or **DDS_RETCODE_NOT_ENABLED** (p. 533).

See also

FooDataWriter::get_key_value (p. 1959)

9.311.2.18 lookup_instance()

```
virtual  DDS_InstanceHandle_t FooDataReader::lookup_instance (
             const  Foo & key_holder )  [pure virtual]
```

Retrieves the instance `handle` that corresponds to an instance `key_holder`.

Useful for keyed data types.

This operation takes as a parameter an instance and returns a handle that can be used in subsequent operations that accept an instance handle as an argument. The instance parameter is only used for the purpose of examining the fields that define the key. If the instance is unknown to the DataReader, or if for any other reason the Service is unable to provide an instance handle, the Service will return the special value **DDS_HANDLE_NIL** (p. 78).

Parameters

key_holder	<<***in***>> *(*p. *433)* a user data type specific key holder.

Returns

the instance handle associated with this instance. If **Foo** (p. 1916) has no key, this method has no effect and returns **DDS_HANDLE_NIL** (p. 78)

9.311.2.19 is_data_consistent()

```
virtual  DDS_ReturnCode_t FooDataReader::is_data_consistent (
              DDS_Boolean & is_data_consistent,
             const TData * sample,
             const  DDS_SampleInfo * sample_info )  [pure virtual]
```

When using Zero Copy transfer over shared memory, checks if the sample has been overwritten by the DataWriter.

When a sample is received via **Zero Copy transfer over shared memory** (p. 73), the sample can be reused by the DataWriter once it is removed from the DataWriter's send queue. Since there is no synchronization between the DataReader and the DataWriter, the sample could be overwritten by the DataWriter before it is processed by the DataReader. The **FooDataReader::is_data_consistent** (p. 1943) operation can be used after processing the sample to check if it was overwritten by the DataWriter.

A precondition for using this operation is to set **DDS_DataWriterShmemRefTransferModeSettings::enable_data_consistency_check** (p. 927) to true.

Warning

This operation cannot be used when the data type is annotated with `@language_binding(FLAT_DATA)`. Reading a FlatData sample delivered with Zero Copy transfer over shared memory while the DataWriter is overwriting it is undefined behavior. An application-level synchronization mechanism is required in this case.

Parameters

is_data_consistent	<<***inout***>> *(*p. *433)* Set to true if the sample is consistent (i.e., the sample has not been overwritten by the DataWriter)
sample	<<***in***>> *(*p. *433)* Sample to be validated
sample_info	<<***in***>> *(*p. *433)* **DDS_SampleInfo** (p. 1344) object received with the sample

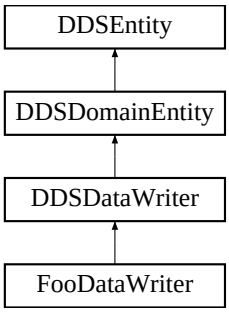
Exceptions

One	of the **Standard Return Codes** (p. 532) or **DDS_RETCODE_PRECONDITION_NOT_MET** (p. 532).

9.312 FooDataWriter Class Reference

*<<**interface**>> (p. 433) <<**generic**>> (p. 433)* User data type specific data writer.

Inheritance diagram for FooDataWriter:



Public Member Functions

- virtual **DDS_InstanceHandle_t register_instance** (const **Foo** &instance_data)=0

  Informs RTI Connext that the application will be modifying a particular instance.
- virtual **DDS_InstanceHandle_t register_instance_w_timestamp** (const **Foo** &instance_data, const **DDS_Time_t** &source_timestamp)=0

  Performs the same functions as register_instance except that the application provides the value for the `source_timestamp`.
- virtual **DDS_InstanceHandle_t register_instance_w_params** (const **Foo** &instance_data, **DDS_WriteParams_t** &params)=0

  *Performs the same function as **FooDataWriter::register_instance** (p. 1946) and **FooDataWriter::register_instance_w_timestamp** (p. 1947) except that it also provides the values contained in* `params`.
- virtual **DDS_ReturnCode_t unregister_instance** (const **Foo** &instance_data, const **DDS_InstanceHandle_t** &handle)=0

  *Reverses the action of **FooDataWriter::register_instance** (p. 1946).*
- virtual **DDS_ReturnCode_t unregister_instance_w_timestamp** (const **Foo** &instance_data, const **DDS_InstanceHandle_t** &handle, const **DDS_Time_t** &source_timestamp)=0

  *Performs the same function as **FooDataWriter::unregister_instance** (p. 1948) except that it also provides the value for the* `source_timestamp`.
- virtual **DDS_ReturnCode_t unregister_instance_w_params** (const **Foo** &instance_data, **DDS_WriteParams_t** &params)=0

  *Performs the same function as **FooDataWriter::unregister_instance** (p. 1948) and **FooDataWriter::unregister_instance_w_timestamp** (p. 1950) except that it also provides the values contained in* `params`.
- virtual **DDS_ReturnCode_t write** (const **Foo** &instance_data, const **DDS_InstanceHandle_t** &handle)=0

  Modifies the value of a data instance.
- virtual **DDS_ReturnCode_t write_w_timestamp** (const **Foo** &instance_data, const **DDS_InstanceHandle_t** &handle, const **DDS_Time_t** &source_timestamp)=0

  *Performs the same function as **FooDataWriter::write** (p. 1951) except that it also provides the value for the* `source_timestamp`.
- virtual **DDS_ReturnCode_t write_w_params** (const **Foo** &instance_data, **DDS_WriteParams_t** &params)=0

  *Performs the same function as **FooDataWriter::write** (p. 1951) and **FooDataWriter::write_w_timestamp** (p. 1955) except that it also provides the values contained in* `params`.

- virtual **DDS_ReturnCode_t dispose** (const **Foo** &instance_data, const **DDS_InstanceHandle_t** &instance_handle)=0

  Requests the middleware to delete the instance.
- virtual **DDS_ReturnCode_t dispose_w_timestamp** (const **Foo** &instance_data, const **DDS_InstanceHandle_t** &instance_handle, const **DDS_Time_t** &source_timestamp)=0

  Performs the same functions as dispose except that the application provides the value for the `source_timestamp` *that is made available to* ***DDSDataReader*** *(*p. *1556) objects by means of the* `source_timestamp` *attribute inside the* ***DDS_SampleInfo*** *(*p. *1344).*
- virtual **DDS_ReturnCode_t dispose_w_params** (const **Foo** &instance_data, **DDS_WriteParams_t** &params)=0

  Performs the same function as ***FooDataWriter::dispose*** *(*p. *1957) and* ***FooDataWriter::dispose_w_timestamp*** *(*p. *1958) except that it also provides the values contained in* `params`*.*
- virtual **DDS_ReturnCode_t get_key_value** (**Foo** &key_holder, const **DDS_InstanceHandle_t** &handle)=0

  Retrieve the instance `key` *that corresponds to an instance* `handle`*.*
- virtual **DDS_InstanceHandle_t lookup_instance** (const **Foo** &key_holder)=0

  Retrieve the instance `handle` *that corresponds to an instance* `key_holder`*.*
- virtual void * **create_data** (const **DDS_TypeAllocationParams_t** &alloc_params=DDS_TYPE_ALLOCATION_PARAMS_DEFAULT)

  Creates a data sample and initializes it.
- virtual **DDS_Boolean delete_data** (void *sample, const **DDS_TypeDeallocationParams_t** &dealloc_params=DDS_TYPE_DEALLOCATION_PARAMS_DEFAULT)

  Destroys a user data type instance.
- virtual **DDS_ReturnCode_t get_loan** (TData *&sample)

  Gets a sample managed by the DataWriter.
- virtual **DDS_ReturnCode_t discard_loan** (TData &sample)

  Returns a loaned sample back to the DataWriter.

Static Public Member Functions

- static **FooDataWriter** * **narrow** (**DDSDataWriter** *writer)

  Narrow the given ***DDSDataWriter*** *(*p. *1589) pointer to a* ***FooDataWriter*** *(*p. *1944) pointer.*

9.312.1 Detailed Description

*<<**interface**>> (*p. *433) <<**generic**>> (*p. *433)* User data type specific data writer.

Defines the user data type specific writer interface generated for each application class.

The concrete user data type writer automatically generated by the implementation is an incarnation of this class.

See also

DDSDataWriter (p. 1589)

Foo (p. 1916)

FooDataReader (p. 1917)

the `Code Generator User's Manual`

9.312.2 Member Function Documentation

9.312.2.1 narrow()

```
static FooDataWriter * FooDataWriter::narrow (
            DDSDataWriter * writer ) [static]
```

Narrow the given **DDSDataWriter** (p. 1589) pointer to a **FooDataWriter** (p. 1944) pointer.

Check if the given `writer` is of type **FooDataWriter** (p. 1944).

Parameters

writer	*<<**in**>> (p. 433)* Base-class **DDSDataWriter** (p. 1589) to be converted to the auto-generated class **FooDataWriter** (p. 1944) that extends **DDSDataWriter** (p. 1589).

Returns

FooDataWriter (p. 1944) if `writer` is of type **Foo** (p. 1916). Return NULL otherwise.

9.312.2.2 register_instance()

```
virtual DDS_InstanceHandle_t FooDataWriter::register_instance (
            const Foo & instance_data ) [pure virtual]
```

Informs RTI Connext that the application will be modifying a particular instance.

This operation is only useful for keyed data types. Using it for non-keyed types causes no effect and returns **DDS_HANDLE_NIL** (p. 78). The operation takes as a parameter an instance (of which only the key value is examined) and returns a `handle` that can be used in successive **write()** (p. 1951) or **dispose()** (p. 1957) operations.

The operation gives RTI Connext an opportunity to pre-configure itself to improve performance.

The use of this operation by an application is optional even for keyed types. If an instance has not been pre-registered, the application can use the special value **DDS_HANDLE_NIL** (p. 78) as the **DDS_InstanceHandle_t** (p. 76) parameter to the write or dispose operation and RTI Connext will auto-register the instance.

For best performance, the operation should be invoked prior to calling any operation that modifies the instance, such as **FooDataWriter::write** (p. 1951), **FooDataWriter::write_w_timestamp** (p. 1955), **FooDataWriter::dispose** (p. 1957) and **FooDataWriter::dispose_w_timestamp** (p. 1958) and the handle used in conjunction with the data for those calls.

When this operation is used, RTI Connext will automatically supply the value of the `source_timestamp` that is used.

This operation may fail and return **DDS_HANDLE_NIL** (p. 78) if **DDS_ResourceLimitsQosPolicy::max_instances** (p. 1311) limit has been exceeded.

The operation is **idempotent**. If it is called for an already registered instance, it just returns the already allocated handle. This may be used to lookup and retrieve the handle allocated to a given instance.

This operation can only be called after **DDSDataWriter** (p. 1589) has been enabled. Otherwise, **DDS_HANDLE_NIL** (p. 78) will be returned.

Parameters

instance_data	<<***in***>> *(*p. *433)* The instance that should be registered. Of this instance, only the fields that represent the key are examined by the function.

Returns

For keyed data type, a handle that can be used in the calls that take a **DDS_InstanceHandle_t** (p. 76), such as write, dispose, unregister_instance, or return **DDS_HANDLE_NIL** (p. 78) on failure. If the instance_data is of a data type that has no keys, this function always returns **DDS_HANDLE_NIL** (p. 78).

See also

FooDataWriter::unregister_instance (p. 1948), **FooDataWriter::get_key_value** (p. 1959), **Relationship between registration, liveliness and ownership** (p. 1225)

9.312.2.3 register_instance_w_timestamp()

```
virtual  DDS_InstanceHandle_t FooDataWriter::register_instance_w_timestamp (
             const  Foo & instance_data,
             const  DDS_Time_t & source_timestamp )  [pure virtual]
```

Performs the same functions as register_instance except that the application provides the value for the source_↩ timestamp.

The provided source_timestamp potentially affects the relative order in which readers observe events from multiple writers. Refer to **DESTINATION_ORDER** (p. 584) QoS policy for details.

This operation may fail and return **DDS_HANDLE_NIL** (p. 78) if **DDS_ResourceLimitsQosPolicy::max_instances** (p. 1311) limit has been exceeded.

This operation can only be called after **DDSDataWriter** (p. 1589) has been enabled. Otherwise, **DDS_HANDLE_NIL** (p. 78) will be returned.

Parameters

instance_data	<<***in***>> *(*p. *433)* The instance that should be registered. Of this instance, only the fields that represent the key are examined by the function.
source_timestamp	<<***in***>> *(*p. *433)* The timestamp value must be greater than or equal to the timestamp value used in the last writer operation (used in a *register*, *unregister*, *dispose*, or *write*, with either the automatically supplied timestamp or the application provided timestamp). This timestamp may potentially affect the order in which readers observe events from multiple writers.

Returns

For keyed data type, return a handle that can be used in the calls that take a **DDS_InstanceHandle_t** (p. 76), such as write, dispose, unregister_instance, or return **DDS_HANDLE_NIL** (p. 78) on failure. If the instance_data is of a data type that has no keys, this function always return **DDS_HANDLE_NIL** (p. 78).

See also

FooDataWriter::unregister_instance (p. 1948), **FooDataWriter::get_key_value** (p. 1959)

9.312.2.4 register_instance_w_params()

```
virtual  DDS_InstanceHandle_t FooDataWriter::register_instance_w_params (
             const  Foo & instance_data,
              DDS_WriteParams_t & params )  [pure virtual]
```

Performs the same function as **FooDataWriter::register_instance** (p. 1946) and **FooDataWriter::register_instance_w_timestamp** (p. 1947) except that it also provides the values contained in `params`.

See also

FooDataWriter::write_w_params (p. 1956)

9.312.2.5 unregister_instance()

```
virtual  DDS_ReturnCode_t FooDataWriter::unregister_instance (
             const  Foo & instance_data,
             const  DDS_InstanceHandle_t & handle )  [pure virtual]
```

Reverses the action of **FooDataWriter::register_instance** (p. 1946).

This operation is useful only for keyed data types. Using it for non-keyed types causes no effect and reports no error. The operation takes as a parameter an instance (of which only the key value is examined) and a handle.

This operation should only be called on an instance that is currently registered. This includes instances that have been auto-registered by calling operations such as write or dispose as described in **FooDataWriter::register_instance** (p. 1946). Otherwise, this operation may fail with **DDS_RETCODE_BAD_PARAMETER** (p. 532).

This only need be called just once per instance, regardless of how many times register_instance was called for that instance.

When this operation is used, RTI Connext will automatically supply the value of the `source_timestamp` that is used.

This operation informs RTI Connext that the **DDSDataWriter** (p. 1589) is no longer going to provide any information about the instance. This operation also indicates that RTI Connext can locally remove all information regarding that

instance. The application should not attempt to use the `handle` previously allocated to that instance after calling this function.

The special value **DDS_HANDLE_NIL** (p. 78) can be used for the parameter `handle`. This indicates that the identity of the instance should be automatically deduced from the `instance_data` (by means of the `key`).

If `handle` is any value other than **DDS_HANDLE_NIL** (p. 78), then it must correspond to an instance that has been registered. If there is no correspondence, the operation will fail with **DDS_RETCODE_BAD_PARAMETER** (p. 532).

RTI Connext will not detect the error when the handle is any value other than **DDS_HANDLE_NIL** (p. 78), corresponds to an instance that has been registered, but does not correspond to the instance deduced from the `instance_data` (by means of the `key`). RTI Connext will treat as if the **unregister_instance()** (p. 1948) operation is for the instance as indicated by the `handle`.

If, after a **FooDataWriter::unregister_instance** (p. 1948), the application wants to modify (**FooDataWriter::write** (p. 1951) or **FooDataWriter::dispose** (p. 1957)) an instance, it has to register it again, or else use the special `handle` value **DDS_HANDLE_NIL** (p. 78).

This operation does not indicate that the instance is deleted (that is the purpose of **FooDataWriter::dispose** (p. 1957)). The operation **FooDataWriter::unregister_instance** (p. 1948) just indicates that the **DDSDataWriter** (p. 1589) no longer has anything to say about the instance. **DDSDataReader** (p. 1556) entities that are reading the instance may receive a sample with **DDS_NOT_ALIVE_NO_WRITERS_INSTANCE_STATE** (p. 166) for the instance, unless there are other **DDSDataWriter** (p. 1589) objects writing that same instance.

DDS_WriterDataLifecycleQosPolicy::autodispose_unregistered_instances (p. 1525) controls whether instances are automatically disposed when they are unregistered.

This operation can affect the ownership of the data instance (see **OWNERSHIP** (p. 612)). If the **DDSDataWriter** (p. 1589) was the exclusive owner of the instance, then calling **unregister_instance()** (p. 1948) will relinquish that ownership.

If **DDS_ReliabilityQosPolicy::kind** (p. 1298) is set to **DDS_RELIABLE_RELIABILITY_QOS** (p. 633) and the unregistration would overflow the resource limits of this writer or of a reader, this operation may block for up to **DDS_ReliabilityQosPolicy::max_blocking_time** (p. 1298); if this writer is still unable to unregister after that period, this method will fail with **DDS_RETCODE_TIMEOUT** (p. 533).

Parameters

instance_data	<<***in***>> *(p. 433)* The instance that should be unregistered. If **Foo** (p. 1916) *has* a key and `instance_handle` is **DDS_HANDLE_NIL** (p. 78), only the fields that represent the key are examined by the function. Otherwise, `instance_data` is not used. If `instance_data` is used, it must represent an instance that has been registerd. Otherwise, this method may fail with **DDS_RETCODE_BAD_PARAMETER** (p. 532) .
handle	<<***in***>> *(p. 433)* represents the `instance` to be unregistered. If **Foo** (p. 1916) *has* a key and `handle` is **DDS_HANDLE_NIL** (p. 78), `handle` is not used and `instance` is deduced from `instance_data`. If **Foo** (p. 1916) has no key, `handle` is not used. If `handle` is used, it must represent an instance that has been registered. Otherwise, this method may fail with **DDS_RETCODE_BAD_PARAMETER** (p. 532).

Exceptions

One	of the **Standard Return Codes** (p. 532), **DDS_RETCODE_TIMEOUT** (p. 533) or **DDS_RETCODE_NOT_ENABLED** (p. 533)

See also

FooDataWriter::register_instance (p. 1946)

FooDataWriter::unregister_instance_w_timestamp (p. 1950)

FooDataWriter::get_key_value (p. 1959)

Relationship between registration, liveliness and ownership (p. 1225)

9.312.2.6 unregister_instance_w_timestamp()

```
virtual  DDS_ReturnCode_t FooDataWriter::unregister_instance_w_timestamp (
             const  Foo & instance_data,
             const  DDS_InstanceHandle_t & handle,
             const  DDS_Time_t & source_timestamp )  [pure virtual]
```

Performs the same function as **FooDataWriter::unregister_instance** (p. 1948) except that it also provides the value for the `source_timestamp`.

The provided `source_timestamp` potentially affects the relative order in which readers observe events from multiple writers. Refer to **DESTINATION_ORDER** (p. 584) QoS policy for details.

The constraints on the values of the `handle` parameter and the corresponding error behavior are the same specified for the **FooDataWriter::unregister_instance** (p. 1948) operation.

This operation may block and may time out (**DDS_RETCODE_TIMEOUT** (p. 533)) under the same circumtances described for the unregister_instance operation.

Parameters

instance_data	<<***in***>> *(p. 433)* The instance that should be unregistered. If **Foo** (p. 1916) *has* a key and `instance_handle` is **DDS_HANDLE_NIL** (p. 78), only the fields that represent the key are examined by the function. Otherwise, `instance_data` is not used. If `instance_data` is used, it must represent an instance that has been registerd. Otherwise, this method may fail with **DDS_RETCODE_BAD_PARAMETER** (p. 532).
handle	<<***in***>> *(p. 433)* represents the `instance` to be unregistered. If **Foo** (p. 1916) *has* a key and `handle` is **DDS_HANDLE_NIL** (p. 78), `handle` is not used and `instance` is deduced from `instance_data`. If **Foo** (p. 1916) has no key, `handle` is not used. If `handle` is used, it must represent an instance that has been registered. Otherwise, this method may fail with **DDS_RETCODE_BAD_PARAMETER** (p. 532).
source_timestamp	<<***in***>> *(p. 433)* The timestamp value must be greater than or equal to the timestamp value used in the last writer operation (used in a *register*, *unregister*, *dispose*, or *write*, with either the automatically supplied timestamp or the application provided timestamp). This timestamp may potentially affect the order in which readers observe events from multiple writers.

Exceptions

One	of the **Standard Return Codes** (p. 532), **DDS_RETCODE_TIMEOUT** (p. 533) or **DDS_RETCODE_NOT_ENABLED** (p. 533).

See also

FooDataWriter::register_instance (p. 1946)

FooDataWriter::unregister_instance (p. 1948)

FooDataWriter::get_key_value (p. 1959)

9.312.2.7 unregister_instance_w_params()

```
virtual  DDS_ReturnCode_t FooDataWriter::unregister_instance_w_params (
             const  Foo & instance_data,
              DDS_WriteParams_t & params )  [pure virtual]
```

Performs the same function as **FooDataWriter::unregister_instance** (p. 1948) and **FooDataWriter::unregister_↩ instance_w_timestamp** (p. 1950) except that it also provides the values contained in `params`.

See also

FooDataWriter::write_w_params (p. 1956)

FooDataWriter::dispose_w_params (p. 1959)

9.312.2.8 write()

```
virtual  DDS_ReturnCode_t FooDataWriter::write (
             const  Foo & instance_data,
             const  DDS_InstanceHandle_t & handle )  [pure virtual]
```

Modifies the value of a data instance.

When this operation is used, RTI Connext will automatically supply the value of the `source_timestamp` that is made available to **DDSDataReader** (p. 1556) objects by means of the `source_timestamp` attribute inside the **DDS_SampleInfo** (p. 1344). (Refer to **DDS_SampleInfo** (p. 1344) and **DESTINATION_ORDER** (p. 584) QoS policy for details).

As a side effect, this operation asserts liveliness on the **DDSDataWriter** (p. 1589) itself, the **DDSPublisher** (p. 1818) and the **DDSDomainParticipant** (p. 1619).

Note that the special value **DDS_HANDLE_NIL** (p. 78) can be used for the parameter `handle`. This indicates the identity of the instance should be automatically deduced from the `instance_data` (by means of the `key`).

If `handle` is any value other than **DDS_HANDLE_NIL** (p. 78), then it must correspond to an instance that has been registered. If there is no correspondence, the operation will fail with **DDS_RETCODE_BAD_PARAMETER** (p. 532).

RTI Connext will not detect the error when the handle is any value other than **DDS_HANDLE_NIL** (p. 78), corresponds to an instance that has been registered, but does not correspond to the instance deduced from the `instance_data`

(by means of the `key`). RTI Connext will treat as if the **write()** (p. 1951) operation is for the instance as indicated by the `handle`.

This operation may block if the **RELIABILITY** (p. 631) `kind` is set to **DDS_RELIABLE_RELIABILITY_QOS** (p. 633) and the modification would cause data to be lost or else cause one of the limits specified in the **RESOURCE_LIMITS** (p. 635) to be exceeded.

This operation will not block when using **DDS_BEST_EFFORT_RELIABILITY_QOS** (p. 633). If you are using BEST↩ _EFFORT Reliability in combination with **DDS_ASYNCHRONOUS_PUBLISH_MODE_QOS** (p. 629), then instead of being blocked, samples that are queued to be sent by the asynchronous publishing thread will be overwritten when the number of DDS samples that are currently queued has reached the `depth` QoS value in the **DDS_HistoryQosPolicy** (p. 1162).

If **DDS_ReliabilityQosPolicy::max_blocking_time** (p. 1298) elapses before the **DDSDataWriter** (p. 1589) can store the modification without exceeding the limits, the operation will fail and return **DDS_RETCODE_TIMEOUT** (p. 533) for KEEP_ALL configurations.

Here is how the write operation behaves when **DDS_KEEP_LAST_HISTORY_QOS** (p. 604) and **DDS_RELIABLE_↩ RELIABILITY_QOS** (p. 633) are used:

- The send window size is determined by the **DDS_RtpsReliableWriterProtocol_t::max_send_window_size** (p. 1333) and **DDS_RtpsReliableWriterProtocol_t::min_send_window_size** (p. 1333) fields in the **DDS_↩ DataWriterProtocolQosPolicy** (p. 892). If a send window is specified (max_send_window_size is not UNLIMITED) and the window is full, the write operation will block until one of the samples in the send window is protocol-acknowledged (ACKed) (1) or until the **DDS_ReliabilityQosPolicy::max_blocking_time** (p. 1298) expires.
- Then, the **DDSDataWriter** (p. 1589) will try to add the new sample to the writer history.
- If the instance associated with the sample is present in the writer history and there are `depth` (in the **HISTORY** (p. 603)) samples in the instance, the DataWriter will replace the oldest sample of that instance independently of that sample's acknowledged status, and the write operation will return **DDS_RETCODE_OK** (p. 532). Otherwise, no sample will be replaced and the write operation will continue.
- If the instance associated with the sample is not present in the writer history and **DDS_ResourceLimitsQos↩ Policy::max_instances** (p. 1311) is exceeded, the DataWriter will try to replace an existing instance (and its samples) according to the value of **DDS_DataWriterResourceLimitsQosPolicy::instance_replacement** (p. 921) (see **DDS_DataWriterResourceLimitsInstanceReplacementKind** (p. 580)).
  - If no instance can be replaced, the write operation returns **DDS_RETCODE_OUT_OF_RESOURCES** (p. 533).
- If **DDS_ResourceLimitsQosPolicy::max_samples** (p. 1311) is exceeded, the DataWriter will try to drop a sample from a different instance as follows:
  - The DataWriter will try first to remove a fully ACKed (2) sample from a different instance 'I' as long as that sample is not the last remaining sample for the instance 'I'. To find this sample, the DataWriter starts iterating from the oldest sample in the writer history to the newest sample.
  - If no such sample is found, the DataWriter will replace the oldest sample in the writer history.
- The sample is added to the writer history, and the write operation returns **DDS_RETCODE_OK** (p. 532).

Here is how the write operation behaves when **DDS_KEEP_ALL_HISTORY_QOS** (p. 604) and **DDS_RELIABLE_↩ RELIABILITY_QOS** (p. 633) are used:

- The send window size is determined by the **DDS_RtpsReliableWriterProtocol_t::max_send_window_size** (p. 1333) and **DDS_RtpsReliableWriterProtocol_t::min_send_window_size** (p. 1333) fields in the **DATA_WRITER_PROTOCOL** (p. 579). If a send window is specified (max_send_window_size is not UNLIMITED) and the window is full, the write operation will block until one of the samples in the send window is protocol-acknowledged (ACKed) (1) or until the **DDS_ReliabilityQosPolicy::max_blocking_time** (p. 1298) expires.
  - If the max_blocking_time expires, the write operation returns **DDS_RETCODE_TIMEOUT** (p. 533).
- When a sample is protocol-ACKed (1) before max_blocking_time expires, the DataWriter will try to add the sample to the writer history as follows:
  - If the instance associated with the sample is not present in the writer history and max_instances is exceeded, the DataWriter will try to replace an existing instance (and its samples) according to the value of **DDS_DataWriterResourceLimitsQosPolicy::instance_replacement** (p. 921) (see **DDS_DataWriterResourceLimitsInstanceReplacementKind** (p. 580)).
    - If no instance can be replaced, the write operation returns **DDS_RETCODE_OUT_OF_RESOURCES** (p. 533).
  - If **DDS_ResourceLimitsQosPolicy::max_samples** (p. 1311) is exceeded, the DataWriter will go through the samples in the order in which they were added, and it will replace the first sample that is fully ACKed (2).
    - If no fully ACKed sample is found, the DataWriter will block (3) until a sample is fully ACKed and can be replaced or **DDS_ReliabilityQosPolicy::max_blocking_time** (p. 1298) expires. If the max_blocking_time expires, the write operation will return **DDS_RETCODE_TIMEOUT** (p. 533).
  - If **DDS_ResourceLimitsQosPolicy::max_samples_per_instance** (p. 1311) is exceeded, the DataWriter will go through the samples of the instance in the order in which they were added, and it will replace the first sample that is fully ACKed.
    - If no fully ACKed sample is found, the DataWriter will block (3) until a sample is fully ACKed and can be replaced or the max_blocking_time expires. If the max_blocking_time expires, the write operation will return **DDS_RETCODE_TIMEOUT** (p. 533).
  - The sample is added to the writer history, and the write operation returns **DDS_RETCODE_OK** (p. 532).

If there are no instance resources left, this operation may fail with **DDS_RETCODE_OUT_OF_RESOURCES** (p. 533). Calling **FooDataWriter::unregister_instance** (p. 1948) may help freeing up some resources.

This operation will fail with **DDS_RETCODE_PRECONDITION_NOT_MET** (p. 532) if the timestamp is less than the timestamp used in the last writer operation (*register*, *unregister*, *dispose*, or *write*, with either the automatically supplied timestamp or the application-provided timestamp).

If an instance obtained from **FooDataWriter::get_loan** (p. 1963) is modified with this operation, then all instances modified thereafter should be from **FooDataWriter::get_loan** (p. 1963).

See **DDS_ReliabilityQosPolicyAcknowledgmentModeKind** (p. 633) for more information on the following notes:

(1) A sample in the writer history is considered "protocol ACKed" when the sample has been individually ACKed at the RTPS protocol level by each one of the DataReaders that matched the DataWriter at the moment the sample was added to the writer queue.

- Late joiners do not change the protocol ACK state of a sample. If a sample is marked as protocol ACKed because it has been acknowledged by all the matching DataReaders and a DataReader joins later on, the historical sample is still considered protocol ACKed even if it has not been received by the late joiner.

- If a sample 'S1' is protocol ACKed and a TopicQuery is received, triggering the publication of 'S1', the sample is still considered protocol ACKed. If a sample 'S1' is not ACKed and a TopicQuery is received triggering the publication of 'S1', the DataWriter will require that both the matching DataReaders on the live RTPS channel and the DataReader on the TopicQuery channel individually protocol ACK the sample in order to consider the sample protocol ACKed.

(2) A sample in the writer history is considered "fully ACKed" when all of the following conditions are met:

- The sample is protocol-ACKed.
- The sample has been "application-level ACKed" by all the DataReaders matching the DataWriter that have their **DDS_ReliabilityQosPolicy::acknowledgment_kind** (p. 1298) set to **DDS_APPLICATION_EXPLICIT_↩ ACKNOWLEDGMENT_MODE** (p. 634) or **DDS_APPLICATION_AUTO_ACKNOWLEDGMENT_MODE** (p. 633). Once the sample is application-level ACKed, it cannot change its status to not ACKed after new DataReaders are matched. (Application-level ACK occurs when the application acknowledges receipt of a sample.)
- If required subscriptions are enabled (see **DDS_AvailabilityQosPolicy** (p. 805)), the sample must also be ACKed by all the required subscriptions configured on the DataWriter.

(3) It is possible within a single call to the write operation for a DataWriter to block both when the send window is full and then again when **DDS_ResourceLimitsQosPolicy::max_samples** (p. 1311) or **DDS_ResourceLimitsQos↩ Policy::max_samples_per_instance** (p. 1311) is exceeded. This can happen because blocking on the send window only considers protocol-ACKed samples, while blocking based on resource limits considers fully-ACKed samples. In any case, the total max blocking time of a single call to the write operation will not exceed **DDS_ReliabilityQosPolicy↩ ::max_blocking_time** (p. 1298).

Parameters

instance_data	*<<**in**>> (p. 433)* The data to write.

Parameters

handle	*<<**in**>> (p. 433)* Either the handle returned by a previous call to **FooDataWriter::register_instance** (p. 1946), or else the special value **DDS_HANDLE_NIL** (p. 78). If **Foo** (p. 1916) *has* a key and `handle` is not **DDS_HANDLE_NIL** (p. 78), `handle` must represent a registered instance of type **Foo** (p. 1916). Otherwise, this method may fail with **DDS_RETCODE_BAD_PARAMETER** (p. 532).

Exceptions

One	of the **Standard Return Codes** (p. 532), **DDS_RETCODE_TIMEOUT** (p. 533), **DDS_RETCODE_PRECONDITION_NOT_MET** (p. 532), **DDS_RETCODE_OUT_OF_RESOURCES** (p. 533), or **DDS_RETCODE_NOT_ENABLED** (p. 533).

MT Safety:

It is UNSAFE to modify `instance_data` before the operation has finished. The operation is otherwise SAFE.

See also

DDSDataReader (p. 1556)

FooDataWriter::write_w_timestamp (p. 1955)

DESTINATION_ORDER (p. 584)

9.312.2.9 write_w_timestamp()

```
virtual DDS_ReturnCode_t FooDataWriter::write_w_timestamp (
            const Foo & instance_data,
            const DDS_InstanceHandle_t & handle,
            const DDS_Time_t & source_timestamp ) [pure virtual]
```

Performs the same function as **FooDataWriter::write** (p. 1951) except that it also provides the value for the `source_timestamp`.

Explicitly provides the timestamp that will be available to the **DDSDataReader** (p. 1556) objects by means of the `source_timestamp` attribute inside the **DDS_SampleInfo** (p. 1344). (Refer to **DDS_SampleInfo** (p. 1344) and **DESTINATION_ORDER** (p. 584) QoS policy for details)

The constraints on the values of the `handle` parameter and the corresponding error behavior are the same specified for the **FooDataWriter::write** (p. 1951) operation.

This operation may block and time out (**DDS_RETCODE_TIMEOUT** (p. 533)) under the same circumtances described for **FooDataWriter::write** (p. 1951).

If there are no instance resources left, this operation may fail with **DDS_RETCODE_OUT_OF_RESOURCES** (p. 533). Calling **FooDataWriter::unregister_instance** (p. 1948) may help free up some resources.

This operation may fail with **DDS_RETCODE_BAD_PARAMETER** (p. 532) under the same circumstances described for the write operation.

Parameters

instance_data	<<***in***>> *(p. 433)* The data to write.
handle	<<***in***>> *(p. 433)* Either the handle returned by a previous call to **FooDataWriter::register_instance** (p. 1946), or else the special value **DDS_HANDLE_NIL** (p. 78). If **Foo** (p. 1916) *has* a key and `handle` is not **DDS_HANDLE_NIL** (p. 78), `handle` must represent a registered instance of type **Foo** (p. 1916). Otherwise, this method may fail with **DDS_RETCODE_BAD_PARAMETER** (p. 532).
source_timestamp	<<***in***>> *(p. 433)* When using **DDS_BY_SOURCE_TIMESTAMP_DESTINATIONORDER_QOS** (p. 585) the timestamp value must be greater than or equal to the timestamp value used in the last writer operation (*register*, *unregister*, *dispose*, or *write*, with either the automatically supplied timestamp or the application-provided timestamp) However, if it is less than the timestamp of the previous operation but the difference is less than the **DDS_DestinationOrderQosPolicy::source_timestamp_tolerance** (p. 933), the timestamp of the previous operation will be used as the source timestamp of this sample. Otherwise, if the difference is greater than **DDS_DestinationOrderQosPolicy::source_timestamp_tolerance** (p. 933), the function will return **DDS_RETCODE_BAD_PARAMETER** (p. 532).

Exceptions

One	of the **Standard Return Codes** (p. 532), **DDS_RETCODE_TIMEOUT** (p. 533), **DDS_RETCODE_OUT_OF_RESOURCES** (p. 533), or **DDS_RETCODE_NOT_ENABLED** (p. 533).

See also

FooDataWriter::write (p. 1951)

DDSDataReader (p. 1556)

DESTINATION_ORDER (p. 584)

9.312.2.10 write_w_params()

```
virtual  DDS_ReturnCode_t FooDataWriter::write_w_params (
             const  Foo & instance_data,
              DDS_WriteParams_t & params )  [pure virtual]
```

Performs the same function as **FooDataWriter::write** (p. 1951) and **FooDataWriter::write_w_timestamp** (p. 1955) except that it also provides the values contained in `params`.

Allows provision of the sample identity, related sample identity, source timestamp, instance handle, and publication priority contained in `params`.

This operation may block and time out (**DDS_RETCODE_TIMEOUT** (p. 533)) under the same circumstances described for **FooDataWriter::write** (p. 1951).

If there are no instance resources left, this operation may fail with **DDS_RETCODE_OUT_OF_RESOURCES** (p. 533). Calling **FooDataWriter::unregister_instance_w_params** (p. 1951) may help free up some resources.

This operation may fail with **DDS_RETCODE_BAD_PARAMETER** (p. 532) under the same circumstances described for the **FooDataWriter::write** (p. 1951).

Parameters

instance_data	<<***in***>> *(p. 433)* The data to write.
params	<<***inout***>> *(p. 433)* The write parameters. Note that this is an inout parameter if you activate **DDS_WriteParams_t::replace_auto** (p. 1520); otherwise it won't be modified.

Exceptions

One	of the **Standard Return Codes** (p. 532), **DDS_RETCODE_TIMEOUT** (p. 533), **DDS_RETCODE_OUT_OF_RESOURCES** (p. 533) or **DDS_RETCODE_NOT_ENABLED** (p. 533).

See also

FooDataWriter::write (p. 1951)

DDSDataReader (p. 1556)

9.312.2.11 dispose()

```
virtual  DDS_ReturnCode_t FooDataWriter::dispose (
             const  Foo & instance_data,
             const  DDS_InstanceHandle_t & instance_handle )  [pure virtual]
```

Requests the middleware to delete the instance.

This operation is useful only for keyed data types. Using it for non-keyed types has no effect and reports no error.

When an instance is disposed, the **DDSDataWriter** (p. 1589) communicates this state change to **DDSDataReader** (p. 1556) objects by propagating a dispose sample. When the instance changes to a disposed state, you can see the state change on the DataReader by looking at **DDS_SampleInfo::instance_state** (p. 1348). Disposed instances have the value **DDS_NOT_ALIVE_DISPOSED_INSTANCE_STATE** (p. 166).

The resources allocated to dispose instances on the DataWriter are not removed by default. The removal of the resources allocated to a dispose instance on the DataWriter queue can be controlled by using the QoS **DDS_Writer↩ DataLifecycleQosPolicy::autopurge_disposed_instances_delay** (p. 1526).

Likewise, on the DataReader, the removal of the resources associated with an instance in the dispose state can be controlled by using the QoS **DDS_ReaderDataLifecycleQosPolicy::autopurge_disposed_instances_delay** (p. 1291).

This operation does not modify the value of the instance. The `instance_data` parameter is passed just for the purposes of identifying the instance.

When this operation is used, RTI Connext will automatically supply the value of the `source_timestamp` that is made available to **DDSDataReader** (p. 1556) objects by means of the source_timestamp attribute inside the **DDS_Sample↩ Info** (p. 1344).

The constraints on the values of the handle parameter and the corresponding error behavior are the same specified for the **FooDataWriter::unregister_instance** (p. 1948) operation.

The special value **DDS_HANDLE_NIL** (p. 78) can be used for the parameter `instance_handle`. This indicates the identity of the instance should be automatically deduced from the `instance_data` (by means of the `key`).

If `instance_handle` is any value other than **DDS_HANDLE_NIL** (p. 78), then it must correspond to an instance that has been registered. If there is no correspondence, the operation will fail with **DDS_RETCODE_BAD_PARAMETER** (p. 532).

RTI Connext will not detect the error when the `instance_handle` is any value other than **DDS_HANDLE_NIL** (p. 78), and the `instance_handle` corresponds to an instance that has been registered but does not correspond to the instance deduced from the `instance_data` (by means of the `key`). In this case, the instance that will be disposed is the instance corresponding to the `instance_handle`, not to the `instance_data`.

This operation may block and time out (**DDS_RETCODE_TIMEOUT** (p. 533)) under the same circumstances described for **FooDataWriter::write** (p. 1951).

If there are no instance resources left, this operation may fail with **DDS_RETCODE_OUT_OF_RESOURCES** (p. 533). Calling **FooDataWriter::unregister_instance** (p. 1948) may help free up some resources.

Parameters

instance_data	<<***in***>> *(p. 433)* The data to dispose. If **Foo** (p. 1916) *has* a key and `instance_handle` is **DDS_HANDLE_NIL** (p. 78), only the fields that represent the key are examined by the function. Otherwise, `instance_data` is not used.
instance_handle	<<***in***>> *(p. 433)* Either the handle returned by a previous call to **FooDataWriter::register_instance** (p. 1946), or else the special value **DDS_HANDLE_NIL** (p. 78). If **Foo** (p. 1916) *has* a key and `instance_handle` is **DDS_HANDLE_NIL** (p. 78), `instance_handle` is not used and it is deduced from `instance_data`. If **Foo** (p. 1916) has no key, `instance_handle` is not used. If `instance_handle` is used, it must represent an instance of type **Foo** (p. 1916) that has been written or registered with this writer. Otherwise, this method fail with **DDS_RETCODE_BAD_PARAMETER** (p. 532). If **Foo** (p. 1916) *has* a key, `instance_handle` cannot be **DDS_HANDLE_NIL** (p. 78) if `instance_data` is NULL. Otherwise, this method will fail with **DDS_RETCODE_BAD_PARAMETER** (p. 532).

Exceptions

One	of the **Standard Return Codes** (p. 532), **DDS_RETCODE_TIMEOUT** (p. 533), **DDS_RETCODE_OUT_OF_RESOURCES** (p. 533) or **DDS_RETCODE_NOT_ENABLED** (p. 533).

See also

FooDataWriter::dispose_w_timestamp (p. 1958)

Relationship between registration, liveliness and ownership (p. 1225)

9.312.2.12 dispose_w_timestamp()

```
virtual  DDS_ReturnCode_t FooDataWriter::dispose_w_timestamp (
             const  Foo & instance_data,
             const  DDS_InstanceHandle_t & instance_handle,
             const  DDS_Time_t & source_timestamp )  [pure virtual]
```

Performs the same functions as dispose except that the application provides the value for the `source_timestamp` that is made available to **DDSDataReader** (p. 1556) objects by means of the `source_timestamp` attribute inside the **DDS_SampleInfo** (p. 1344).

The constraints on the values of the `handle` parameter and the corresponding error behavior are the same specified for the **FooDataWriter::dispose** (p. 1957) operation.

This operation may block and time out (**DDS_RETCODE_TIMEOUT** (p. 533)) under the same circumtances described for **FooDataWriter::write** (p. 1951).

If there are no instance resources left, this operation may fail with **DDS_RETCODE_OUT_OF_RESOURCES** (p. 533). Calling **FooDataWriter::unregister_instance** (p. 1948) may help freeing up some resources.

Parameters

instance_data	<<***in***>> *(p. 433)* The data to dispose. If **Foo** (p. 1916) *has* a key and `instance_handle` is **DDS_HANDLE_NIL** (p. 78), only the fields that represent the key are examined by the function. Otherwise, `instance_data` is not used.
instance_handle	<<***in***>> *(p. 433)* Either the handle returned by a previous call to **FooDataWriter::register_instance** (p. 1946), or else the special value **DDS_HANDLE_NIL** (p. 78). If **Foo** (p. 1916) *has* a key and `handle` is not **DDS_HANDLE_NIL** (p. 78), `handle` must represent a registered instance of type **Foo** (p. 1916). Otherwise, this method may fail with **DDS_RETCODE_BAD_PARAMETER** (p. 532).
source_timestamp	<<***in***>> *(p. 433)* The timestamp value must be greater than or equal to the timestamp value used in the last writer operation (used in a *register*, *unregister*, *dispose*, or *write*, with either the automatically supplied timestamp or the application provided timestamp). This timestamp may potentially affect the order in which readers observe events from multiple writers. This timestamp will be available to the **DDSDataReader** (p. 1556) objects by means of the `source_timestamp` attribute inside the **DDS_SampleInfo** (p. 1344).

Exceptions

One	of the **Standard Return Codes** (p. 532), **DDS_RETCODE_TIMEOUT** (p. 533), **DDS_RETCODE_OUT_OF_RESOURCES** (p. 533) or **DDS_RETCODE_NOT_ENABLED** (p. 533).

See also

FooDataWriter::dispose (p. 1957)

9.312.2.13 dispose_w_params()

```
virtual  DDS_ReturnCode_t FooDataWriter::dispose_w_params (
             const  Foo & instance_data,
              DDS_WriteParams_t & params )  [pure virtual]
```

Performs the same function as **FooDataWriter::dispose** (p. 1957) and **FooDataWriter::dispose_w_timestamp** (p. 1958) except that it also provides the values contained in `params`.

See also

FooDataWriter::write_w_params (p. 1956)

9.312.2.14 get_key_value()

```
virtual  DDS_ReturnCode_t FooDataWriter::get_key_value (
              Foo & key_holder,
             const  DDS_InstanceHandle_t & handle )  [pure virtual]
```

Retrieve the instance `key` that corresponds to an instance `handle`.

Useful for keyed data types.

The operation will only fill the fields that form the `key` inside the `key_holder` instance. If the type has no keys, this method has no effect and exits with no error.

For keyed data types, this operation may fail with **DDS_RETCODE_BAD_PARAMETER** (p. 532) if the `handle` does not correspond to an existing data-object known to the **DDSDataWriter** (p. 1589).

Parameters

key_holder	<<***inout***>> *(p. 433)* a user data type specific key holder, whose `key` fields are filled by this operation. If **Foo** (p. 1916) has no key, this method has no effect.

Parameters

handle	<<***in***>> *(p. 433)* the `instance` whose key is to be retrieved. If **Foo** (p. 1916) *has* a key, `handle` must represent a registered instance of type **Foo** (p. 1916). Otherwise, this method will fail with **DDS_RETCODE_BAD_PARAMETER** (p. 532). If **Foo** (p. 1916) *has* a key and `handle` is **DDS_HANDLE_NIL** (p. 78), this method will fail with **DDS_RETCODE_BAD_PARAMETER** (p. 532).

Exceptions

One	of the **Standard Return Codes** (p. 532) or **DDS_RETCODE_NOT_ENABLED** (p. 533).

See also

FooDataReader::get_key_value (p. 1941)

9.312.2.15 lookup_instance()

```
virtual  DDS_InstanceHandle_t FooDataWriter::lookup_instance (
            const  Foo & key_holder )  [pure virtual]
```

Retrieve the instance `handle` that corresponds to an instance `key_holder`.

Useful for keyed data types.

This operation takes as a parameter an instance and returns a handle that can be used in subsequent operations that accept an instance handle as an argument. The instance parameter is only used for the purpose of examining the fields that define the key. This operation does not register the instance in question. If the instance has not been previously registered, or if for any other reason RTI Connext is unable to provide an instance handle, RTI Connext will return the special value HANDLE_NIL.

Parameters

key_holder	<<***in***>> *(*p. *433)* a user data type specific key holder.

Returns

the instance handle associated with this instance. If **Foo** (p. 1916) has no key, this method has no effect and returns **DDS_HANDLE_NIL** (p. 78)

9.312.2.16 create_data()

```
virtual void * FooDataWriter::create_data (
            const  DDS_TypeAllocationParams_t & alloc_params = DDS_TYPE_ALLOCATION_PARAMS_DEFAULT
)  [inline], [virtual]
```

Creates a data sample and initializes it.

The behavior of this API is identical to **FooTypeSupport::create_data** (p. 1983).

Parameters

alloc_params	<<***in***>> *(*p. *433)* Whether or not to recursively allocate pointers and/or optional members

Returns

Newly created data type, or NULL on failure.

See also

FooDataWriter::delete_data (p. 1962)

9.312.2.17 delete_data()

```
virtual  DDS_Boolean FooDataWriter::delete_data (
            void * sample,
            const  DDS_TypeDeallocationParams_t & dealloc_params = DDS_TYPE_DEALLOCATION_PARAMS↩
_DEFAULT )  [inline], [virtual]
```

Destroys a user data type instance.

The behavior of this API is identical to **FooTypeSupport::delete_data** (p. 1984).

Parameters

sample	<<***in***>> *(p. 433)* Cannot be NULL.
dealloc_params	<<***in***>> *(p. 433)* Whether or not to destroy pointers and/or optional members. By default they are destroyed (deleted).

Returns

DDS_BOOLEAN_TRUE (p. 513) on success.

See also

FooDataWriter::create_data (p. 1962)

9.312.2.18 get_loan()

```
virtual  DDS_ReturnCode_t FooDataWriter::get_loan (
             TData *& sample )  [inline], [virtual]
```

Gets a sample managed by the DataWriter.

This operation is supported while using **Zero Copy transfer** (p. 73) over shared memory" or \ref RTIFlatDataModule "FlatData language binding". For FlatData types, this function should be used directly if the type is final; if the type is mutable, **rti::flat::build_data()** (p. 768) should be used to obtain a sample builder to work with the loaned sample.

The loaned sample is obtained from a DataWriter-managed sample pool and is uninitialized by default. An initialized sample can be obtained by setting **DDS_DataWriterResourceLimitsQosPolicy::initialize_writer_loaned_sample** (p. 925) to **DDS_BOOLEAN_TRUE** (p. 513). The **DDS_DataWriterResourceLimitsQosPolicy::writer_loaned_sample_allocation** (p. 925) settings can be used to configure the DataWriter-managed sample pool.

FooDataWriter::get_loan (p. 1963) fails with **DDS_RETCODE_OUT_OF_RESOURCES** (p. 533) if **DDS_AllocationSettings_t::max_count** (p. 797) samples have been loaned, and none of those samples has been written with **FooDataWriter::write** (p. 1951) or discarded via **FooDataWriter::discard_loan** (p. 1964).

Samples returned from **FooDataWriter::get_loan** (p. 1963) have an associated state. Due to the optimized nature of the write operation while using Zero Copy transfer over shared memory or FlatData language binding, this sample state is used to control when a sample is available for reuse after the write operation. The possible sample states are free, allocated, removed or serialized. A sample that has never been allocated is "free". **FooDataWriter::get_loan** (p. 1963) takes a "free" or "removed" sample and makes it "allocated". When a sample is written, its state transitions from "allocated" to "serialized", and the DataWriter takes responsibility for returning the sample back to its sample pool. The sample remains in the "serialized" state until it is removed from the DataWriter queue. For a reliable DataWriter, the sample is removed from the DataWriter's queue when the sample is acknowledged by all DataReaders. For a best-effort DataWriter, the sample is removed from the queue immediately after the write operation. After the sample is removed from the DataWriter queue, the sample is put back into the sample pool, and its state transitions from "serialized" to "removed". At this time, a new call to **FooDataWriter::get_loan** (p. 1963) may return the same sample.

A loaned sample should not be reused to write a new value after the first write operation. Instead, a new sample from **FooDataWriter::get_loan** (p. 1963) should be used to write the new value. A loaned sample that has not been written

can be returned to the DataWriter's sample pool by using **FooDataWriter::discard_loan** (p. 1964). If the write operation fails, then the sample can be used again with a write or discard_loan operation. Disposing or unregistering an instance with loaned samples follows the same pattern. A loaned sample used successfully with a dispose or unregister operation cannot be used again. But if the dispose or unregister operation fails, the sample is available for reuse.

A DataWriter cannot write managed samples (created with get_loan) and unmanaged samples (created in any other way) at the same time. The first call to get_loan automatically prepares this DataWriter to work with managed samples. Calls to get_loan will fail with **DDS_RETCODE_PRECONDITION_NOT_MET** (p. 532) if an unmanaged sample was written with this DataWriter earlier. Similarly, **FooDataWriter::write** (p. 1951) will fail to write an unmanaged sample if get_loan was called.

Parameters

sample	<<***inout***>> *(p. 433)* reference to a user data type pointer. The loaned sample is returned via this sample.

Exceptions

One	of the **Standard Return Codes** (p. 532), **DDS_RETCODE_OUT_OF_RESOURCES** (p. 533) or **DDS_RETCODE_PRECONDITION_NOT_MET** (p. 532).

See also

FooDataWriter::discard_loan (p. 1964)

9.312.2.19 discard_loan()

```
virtual  DDS_ReturnCode_t FooDataWriter::discard_loan (
             TData & sample )  [inline], [virtual]
```

Returns a loaned sample back to the DataWriter.

This operation is supported while using **Zero Copy transfer** (p. 73) over shared memory" or the **FlatData language binding** (p. 776).

A loaned sample that hasn't been written can be returned to the DataWriter with this operation.

Parameters

sample	<<***in***>> *(p. 433)* loaned sample to be discarded.

Exceptions

One	of the **Standard Return Codes** (p. 532) or **DDS_RETCODE_NOT_ENABLED** (p. 533).

See also

FooDataWriter::get_loan (p. 1963)

9.313 FooSeq Struct Reference

*<<**interface**>> (p. 433) <<**generic**>> (p. 433)* A type-safe, ordered collection of elements. The type of these elements is referred to in this documentation as **`Foo`** (p. 1916).

Public Member Functions

- **FooSeq** & **operator=** (const struct **FooSeq** &src_seq)

  Copy elements from another sequence, resizing the sequence if necessary.
- bool **copy_no_alloc** (const struct **FooSeq** &src_seq)

  Copy elements from another sequence, only if the destination sequence has enough capacity.
- bool **from_array** (const **Foo** array[ ], **DDS_Long** **length**)

  Copy elements from an array of elements, resizing the sequence if necessary. The original contents of the sequence (if any) are replaced.
- bool **to_array** (**Foo** array[ ], **DDS_Long** **length**)

  Copy elements to an array of elements. The original contents of the array (if any) are replaced.
- **Foo** & **operator[ ]** (**DDS_Long** i)

  Set the `i-th` element of the sequence.
- const **Foo** & **operator[ ]** (**DDS_Long** i) const

  Get the `i-th` element for a `const` sequence.
- **DDS_Long** **length** () const

  Get the logical length of this sequence.
- bool **length** (**DDS_Long** new_length)

  Set the sequence to the desired length, and resize the sequence if necessary.
- bool **ensure_length** (**DDS_Long** **length**, **DDS_Long** max)

  Set the sequence to the desired length, and resize the sequence if necessary.
- **DDS_Long** **maximum** () const

  Get the current maximum number of elements that can be stored in this sequence.
- bool **maximum** (**DDS_Long** new_max)

  Resize this sequence to a new desired maximum.
- bool **loan_contiguous** (**Foo** *buffer, **DDS_Long** new_length, **DDS_Long** new_max)

  Loan a contiguous buffer to this sequence.
- bool **loan_discontiguous** (**Foo** **buffer, **DDS_Long** new_length, **DDS_Long** new_max)

  Loan a discontiguous buffer to this sequence.
- bool **unloan** ()

  Return the loaned buffer in the sequence and set the maximum to 0.
- **Foo** * **get_contiguous_buffer** () const

  Return the contiguous buffer of the sequence.
- **Foo** ** **get_discontiguous_buffer** () const

  Return the discontiguous buffer of the sequence.
- bool **has_ownership** ()

  Return the value of the owned flag.

- ~**FooSeq** ()

  Deallocate this sequence's buffer.
- **FooSeq** (**DDS_Long** new_max=0)

  Create a sequence with the given maximum.
- **FooSeq** (const struct **FooSeq** &foo_seq)

  Create a sequence by copying from an existing sequence.

9.313.1 Detailed Description

<<***interface***>> *(*p. *433)* <<***generic***>> *(*p. *433)* A type-safe, ordered collection of elements. The type of these elements is referred to in this documentation as **Foo** (p. 1916).

For users who define data types in OMG IDL, this type corresponds to the IDL express sequence<**Foo** (p. 1916)>.

For any user-data type **Foo** (p. 1916) that an application defines for the purpose of data-distribution with RTI Connext, a **FooSeq** (p. 1965) is generated. The sequence offers a subset of the methods defined by the standard OMG IDL to C++ mapping for sequences. We refer to an IDL sequence<**Foo** (p. 1916)> as **FooSeq** (p. 1965).

The state of a sequence is described by the properties 'maximum', 'length' and 'owned'.

- The 'maximum' represents the size of the underlying buffer; this is the maximum number of elements it can possibly hold. It is returned by the **FooSeq::maximum()** (p. 1975) operation.

- The 'length' represents the actual number of elements it currently holds. It is returned by the **FooSeq::length()** (p. 1973) operation.

- The 'owned' flag represents whether the sequence owns the underlying buffer. It is returned by the **FooSeq**↩**::has_ownership** (p. 1979) operation. If the sequence does not own the underlying buffer, the underlying buffer is loaned from somewhere else. This flag influences the lifecycle of the sequence and what operations are allowed on it. The general guidelines are provided below and more details are described in detail as pre-conditions and post-conditions of each of the sequence's operations:
  - If owned == **DDS_BOOLEAN_TRUE** (p. 513), the sequence has ownership on the buffer. It is then responsible for destroying the buffer when the sequence is destroyed.
  - If the owned == **DDS_BOOLEAN_FALSE** (p. 513), the sequence does not have ownership on the buffer. This implies that the sequence is loaning the buffer. The sequence cannot be destroyed until the loan is returned.
  - A sequence with a zero maximum always has owned == **DDS_BOOLEAN_TRUE** (p. 513)

See also

FooDataWriter (p. 1944), **FooDataReader** (p. 1917), **FooTypeSupport** (p. 1979), the Code Generator User's Manual

9.313.2 Constructor & Destructor Documentation

9.313.2.1 ~FooSeq()

```
FooSeq::~FooSeq ( )
```

Deallocate this sequence's buffer.

Precondition

(owned == **DDS_BOOLEAN_TRUE** (p. 513)). If this precondition is not met, no memory will be freed and an error will be logged.

Postcondition

maximum == 0 and the underlying buffer is freed.

See also

FooSeq::maximum() (p. 1975), **FooSeq::unloan** (p. 1977)

9.313.2.2 FooSeq() [1/2]

```
FooSeq::FooSeq (
            DDS_Long new_max = 0 )
```

Create a sequence with the given maximum.

This is a constructor for the sequence. The constructor will automatically allocate memory to hold new_max elements of type **Foo** (p. 1916).

This constructor will be used when the application creates a sequence using one of the following:

```
FooSeq mySeq(5);

// or

FooSeq mySeq;

// or

FooSeq* mySeqPtr = new FooSeq(5);
```

Postcondition

maximum == new_max

length == 0

owned == **DDS_BOOLEAN_TRUE** (p. 513),

Parameters

new_max	Must be >= 0. Otherwise the sequence will be initialized to a new_max=0.

9.313.2.3 FooSeq() [2/2]

```
FooSeq::FooSeq (
            const struct  FooSeq & foo_seq )
```

Create a sequence by copying from an existing sequence.

This is a constructor for the sequence. The constructor will automatically allocate memory to hold foo_seq::maximum() elements of type **Foo** (p. 1916) and will copy the current contents of foo_seq into the new sequence.

This constructor will be used when the application creates a sequence using one of the following:

```
FooSeq mySeq(foo_seq);

// or

FooSeq mySeq = foo_seq;

// or

FooSeq *mySeqPtr = new FooSeq(foo_seq);
```

Postcondition

this::maximum == foo_seq::maximum

this::length == foo_seq::length

this[i] == foo_seq[i] for 0 <= i < foo_seq::length

this::owned == **DDS_BOOLEAN_TRUE** (p. 513)

Note

If the pre-conditions are not met, or an error occurs during copying, the constructor will initialize the new sequence to a maximum of zero.

9.313.3 Member Function Documentation

9.313.3.1 operator=()

```
FooSeq & FooSeq::operator= (
            const struct  FooSeq & src_seq )
```

Copy elements from another sequence, resizing the sequence if necessary.

This method invokes **FooSeq::copy_no_alloc** (p. 1970) after ensuring that the sequence has enough capacity to hold the elements to be copied.

This operator is invoked when the following expression appears in the code:
`target_seq = src_seq`

Important: This method *will* allocate memory if this::maximum $<$ src_seq::length.

Therefore, to programatically detect the successful completion of the operator it is recommended that the application first sets the length of this sequence to zero, makes the assignment, and then checks that the length of this sequence matches that of src_seq.

Parameters

src_seq	<<***in***>> *(p. 433)* the sequence from which to copy

See also

FooSeq::copy_no_alloc (p. 1970)

9.313.3.2 copy_no_alloc()

```
bool FooSeq::copy_no_alloc (
            const struct  FooSeq & src_seq )
```

Copy elements from another sequence, only if the destination sequence has enough capacity.

Fill the elements in this sequence by copying the corresponding elements in `src_seq`. The original contents in this sequence are replaced via the element assignment operation (Foo_copy() function). By default, elements are discarded; 'delete' is not invoked on the discarded elements.

Precondition

this::maximum >= src_seq::length

this::owned == **DDS_BOOLEAN_TRUE** (p. 513)

Postcondition

this::length == src_seq::length

this[i] == src_seq[i] for 0 <= i < target_seq::length

this::owned == **DDS_BOOLEAN_TRUE** (p. 513)

Parameters

src_seq	<<***in***>> *(p. 433)* the sequence from which to copy

Returns

DDS_BOOLEAN_TRUE (p. 513) if the sequence was successfully copied; **DDS_BOOLEAN_FALSE** (p. 513) otherwise.

Note

If the pre-conditions are not met, the operator will print a message to stdout and leave this sequence unchanged.

See also

FooSeq::operator= (p. 1968)

9.313.3.3 from_array()

```
bool FooSeq::from_array (
             const  Foo array[],
              DDS_Long length )
```

Copy elements from an array of elements, resizing the sequence if necessary. The original contents of the sequence (if any) are replaced.

Fill the elements in this sequence by copying the corresponding elements in `array`. The original contents in this sequence are replaced via the element assignment operation (Foo_copy() function). By default, elements are discarded; 'delete' is not invoked on the discarded elements.

Precondition

this::owned == **DDS_BOOLEAN_TRUE** (p. 513)

Postcondition

this::length == `length`

this[i] == array[i] for 0 <= i < `length`

this::owned == **DDS_BOOLEAN_TRUE** (p. 513)

Parameters

array	<<***in***>> *(p. 433)* The array of elements to be copy elements from
length	<<***in***>> *(p. 433)* The length of the array.

Returns

DDS_BOOLEAN_TRUE (p. 513) if the array was successfully copied; **DDS_BOOLEAN_FALSE** (p. 513) otherwise.

Note

If the pre-conditions are not met, the method will print a message to stdout and leave this sequence unchanged.

9.313.3.4 to_array()

```
bool FooSeq::to_array (
                Foo array[],
                DDS_Long length )
```

Copy elements to an array of elements. The original contents of the array (if any) are replaced.

Copy the elements of this sequence to the corresponding elements in the array. The original contents of the array are replaced via the element assignment operation (Foo_copy() function). By default, elements are discarded; 'delete' is not invoked on the discarded elements.

Parameters

array	<<***in***>> *(p. 433)* The array of elements to be filled with elements from this sequence
length	<<***in***>> *(p. 433)* The number of elements to be copied.

Returns

DDS_BOOLEAN_TRUE (p. 513) if the elements of the sequence were successfully copied; **DDS_BOOLEAN_↩ FALSE** (p. 513) otherwise.

9.313.3.5 operator[]() [1/2]

```
Foo & FooSeq::operator[] (
                DDS_Long i )
```

Set the `i-th` element of the sequence.

This is the operator that is invoked when the application indexes into a non- `const` sequence:

```
myElement = mySequence[i];

mySequence[i] = myElement;
```

Note that a *reference* to the `i-th` element is returned (and not a copy).

Parameters

i	index of element to access, must be >= 0 and less than **FooSeq::length()** (p. 1973)

Returns

the `i-th` element

9.313.3.6 operator[]() [2/2]

```
const  Foo & FooSeq::operator[] (
               DDS_Long i ) const
```

Get the i-th element for a const sequence.

This is the operator that is invoked when the application indexes into a const sequence:
```
myElement = mySequence[i];
```

Note that a *reference* to the i-th element is returned (and not a copy).

Parameters

i	index of element to access, must be >= 0 and less than **FooSeq::length()** (p. 1973)

Returns

the i-th element

9.313.3.7 length() [1/2]

```
DDS_Long FooSeq::length ( ) const
```

Get the logical length of this sequence.

Get the length that was last set, or zero if the length has never been set.

Returns

the length of the sequence

9.313.3.8 length() [2/2]

```
bool FooSeq::length (
               DDS_Long new_length )
```

Set the sequence to the desired length, and resize the sequence if necessary.

If the current maximum is greater than the desired length, then sequence is not resized.

Otherwise, if this sequence owns its buffer, the sequence is resized to the new length by freeing and re-allocating the buffer. However, if the sequence does not own its buffer, this operation will fail.

For sequences that are part of a type declared in IDL, the length must not exceed the maximum established for the sequence in the IDL.

Parameters

new_length	the new desired length. This value must be non-negative. Must be >= 0.

Returns

DDS_BOOLEAN_TRUE (p. 513) on success or **DDS_BOOLEAN_FALSE** (p. 513) on failure

9.313.3.9 ensure_length()

```
bool FooSeq::ensure_length (
              DDS_Long length,
              DDS_Long max )
```

Set the sequence to the desired length, and resize the sequence if necessary.

If the current maximum is greater than the new length, then the sequence is not resized.

Otherwise, if this sequence owns its buffer, the sequence is resized to the new maximum by freeing and re-allocating the buffer, and the length is set to the new length. However, if the sequence does not own its buffer, this operation will fail.

This function allows user to avoid unnecessary buffer re-allocation.

Precondition

`length` <= `max`

`max` <= maximum size for IDL bounded sequences

owned == **DDS_BOOLEAN_TRUE** (p. 513) if sequence needs to be resized

Postcondition

length == `length`

maximum == `max` if resized

Parameters

length	<<***in***>> *(p. 433)* The new length that should be set. Must be >= 0.
max	<<***in***>> *(p. 433)* If sequence need to be resized, this is the maximum that should be set. `max` >= `length`

Returns

DDS_BOOLEAN_TRUE (p. 513) on success, **DDS_BOOLEAN_FALSE** (p. 513) if the preconditions are not met. In that case the sequence is not modified.

9.313.3.10 maximum() [1/2]

```
DDS_Long FooSeq::maximum ( ) const
```

Get the current maximum number of elements that can be stored in this sequence.

The `maximum` of the sequence represents the maximum number of elements that the underlying buffer can hold. It does not represent the current number of elements.

The `maximum` is a non-negative number. It is initialized when the sequence is first created.

`maximum` can only be changed with the **FooSeq::maximum(DDS_Long)** (p. 1975) operation.

Returns

the current maximum of the sequence.

See also

FooSeq::length() (p. 1973)

9.313.3.11 maximum() [2/2]

```
bool FooSeq::maximum (
            DDS_Long new_max )
```

Resize this sequence to a new desired maximum.

This operation does nothing if the new desired maximum matches the current maximum.

If this sequence owns its buffer and the new maximum is not equal to the old maximum, then the existing buffer will be freed and re-allocated.

Precondition

owned == **DDS_BOOLEAN_TRUE** (p. 513)

`new_max` <= maximum size for IDL bounded sequences.

Postcondition

owned == **DDS_BOOLEAN_TRUE** (p. 513)

length == MINIMUM(original length, new_max)

Parameters

new_max	Must be >= 0.

Returns

DDS_BOOLEAN_TRUE (p. 513) on success, **DDS_BOOLEAN_FALSE** (p. 513) if the preconditions are not met. In that case the sequence is not modified.

9.313.3.12 loan_contiguous()

```
bool FooSeq::loan_contiguous (
                Foo * buffer,
                DDS_Long new_length,
                DDS_Long new_max )
```

Loan a contiguous buffer to this sequence.

This operation changes the `owned` flag of the sequence to **DDS_BOOLEAN_FALSE** (p. 513) and also sets the underlying buffer used by the sequence. See the `User's Manual` for more information about sequences and memory ownership.

Use this method if you want to manage the memory used by the sequence yourself. You must provide an array of elements and integers indicating how many elements are allocated in that array (i.e. the maximum) and how many elements are valid (i.e. the length). The sequence will subsequently use the memory you provide and will not permit it to be freed by a call to **FooSeq::maximum(DDS_Long)** (p. 1975).

Once you have loaned a buffer to a sequence, make sure that you don't free it before calling **FooSeq::unloan** (p. 1977): the next time you access the sequence, you will be accessing freed memory!

You can use this method to wrap stack memory with a sequence interface, thereby avoiding dynamic memory allocation. Create a **FooSeq** (p. 1965) and an array of type **Foo** (p. 1916) and then loan the array to the sequence:

```
::Foo fooArray[10];

::FooSeq fooSeq;

fooSeq.loan_contiguous(fooArray, 0, 10);
```

By default, a sequence you create owns its memory unless you explicitly loan memory of your own to it. In a very few cases, RTI Connext will return a sequence to you that has a loan; those cases are documented as such. For example, if you call **FooDataReader::read** (p. 1919) or **FooDataReader::take** (p. 1920) and pass in sequences with no loan and no memory allocated, RTI Connext will loan memory to your sequences which must be unloaned with **FooDataReader::return_loan** (p. 1940). See the documentation of those methods for more information.

Precondition

FooSeq::maximum() (p. 1975) == 0; i.e. the sequence has no memory allocated to it.

FooSeq::has_ownership (p. 1979) == **DDS_BOOLEAN_TRUE** (p. 513); i.e. the sequence does not already have an outstanding loan

Postcondition

The sequence will store its elements in the buffer provided.

FooSeq::has_ownership (p. 1979) == **DDS_BOOLEAN_FALSE** (p. 513)

FooSeq::length() (p. 1973) == new_length

FooSeq::maximum() (p. 1975) == new_max

Parameters

buffer	The new buffer that the sequence will use. Must point to enough memory to hold new_max elements of type **Foo** (p. 1916). It may be NULL if new_max == 0.
new_length	The desired new length for the sequence. It must be the case that that 0 <= new_length <= new_max.
new_max	The allocated number of elements that could fit in the loaned buffer.

Returns

DDS_BOOLEAN_TRUE (p. 513) if `buffer` is successfully loaned to this sequence or **DDS_BOOLEAN_FALSE** (p. 513) otherwise. Failure only occurs due to failing to meet the pre-conditions. Upon failure the sequence remains unmodified.

See also

FooSeq::unloan (p. 1977) , **FooSeq::loan_discontiguous** (p. 1977)

9.313.3.13 loan_discontiguous()

```
bool FooSeq::loan_discontiguous (
              Foo ** buffer,
              DDS_Long new_length,
              DDS_Long new_max )
```

Loan a discontiguous buffer to this sequence.

This method is exactly like **FooSeq::loan_contiguous** (p. 1976) except that the buffer loaned is an array of **Foo** (p. 1916) pointers, not an array of **Foo** (p. 1916).

Parameters

buffer	The new buffer that the sequence will use. Must point to enough memory to hold new_max elements of type Foo*. It may be NULL if new_max == 0.
new_length	The desired new length for the sequence. It must be the case that that 0 <= new_length <= new_max.
new_max	The allocated number of elements that could fit in the loaned buffer.

See also

FooSeq::unloan (p. 1977), **FooSeq::loan_contiguous** (p. 1976)

9.313.3.14 unloan()

```
bool FooSeq::unloan ( )
```

Return the loaned buffer in the sequence and set the maximum to 0.

This method affects only the state of this sequence; it does not change the contents of the buffer in any way.

Only the user who originally loaned a buffer should return that loan, as the user may have dependencies on that memory known only to them. Unloaning someone else's buffer may cause unspecified problems. For example, suppose a sequence is loaning memory from a custom memory pool. A user of the sequence likely has no way to release the memory back into the pool, so unloaning the sequence buffer would result in a resource leak. If the user were to then re-loan a different buffer, the original creator of the sequence would have no way to discover, when freeing the sequence, that the loan no longer referred to its own memory and would thus not free the user's memory properly, exacerbating the situation and leading to undefined behavior.

Precondition

owned == **DDS_BOOLEAN_FALSE** (p. 513)

Postcondition

owned == **DDS_BOOLEAN_TRUE** (p. 513)

maximum == 0

Returns

DDS_BOOLEAN_TRUE (p. 513) if the preconditions were met. Otherwise **DDS_BOOLEAN_FALSE** (p. 513). The function only fails if the pre-conditions are not met, in which case it leaves the sequence unmodified.

See also

FooSeq::loan_contiguous (p. 1976), **FooSeq::loan_discontiguous** (p. 1977), **FooSeq::maximum(DDS_↩ Long)** (p. 1975)

9.313.3.15 get_contiguous_buffer()

```
Foo * FooSeq::get_contiguous_buffer ( ) const
```

Return the contiguous buffer of the sequence.

Get the underlying buffer where contiguous elements of the sequence are stored. The size of the buffer matches the maximum of the sequence, but only the elements up to the **FooSeq::length()** (p. 1973) of the sequence are valid.

This method provides almost no encapsulation of the sequence's underlying implementation. Certain operations, such as **FooSeq::maximum(DDS_Long)** (p. 1975), may render the buffer invalid. In light of these caveats, this operation should be used with care.

Returns

buffer that stores contiguous elements in sequence.

9.313.3.16 get_discontiguous_buffer()

```
Foo ** FooSeq::get_discontiguous_buffer ( ) const
```

Return the discontiguous buffer of the sequence.

This operation returns the underlying buffer where discontiguous elements of the sequence are stored. The size of the buffer matches the maximum of this sequence, but only the elements up to the **FooSeq::length()** (p. 1973) of the sequence are valid.

The same caveats apply to this method as to **FooSeq::get_contiguous_buffer** (p. 1978).

The sequence will dereference pointers in the discontiguous buffer to provide access to its elements by value in C and by reference in C++. If you access the discontiguous buffer directly by means of this method, do not store any NULL values into it, as acessing those values will result in a segmentation fault.

Returns

buffer that stores discontiguous elements in sequence.

9.313.3.17 has_ownership()

```
bool FooSeq::has_ownership ( )
```

Return the value of the owned flag.

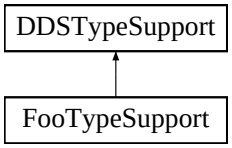
Returns

DDS_BOOLEAN_TRUE (p. 513) if sequence owns the underlying buffer, or **DDS_BOOLEAN_FALSE** (p. 513) if it has an outstanding loan.

9.314 FooTypeSupport Class Reference

*<<**interface**>> (p. 433) <<**generic**>> (p. 433)* User data type specific interface.

Inheritance diagram for FooTypeSupport:



Static Public Member Functions

- static **DDS_ReturnCode_t register_type** (**DDSDomainParticipant** *participant, const char *type_name)

  Allows an application to communicate to RTI Connext the existence of a data type.
- static **DDS_ReturnCode_t unregister_type** (**DDSDomainParticipant** *participant, const char *type_name)

  <<***extension***>> *(p. 432) Allows an application to unregister a data type from RTI Connext. After calling unregister_type, no further communication using that type is possible.*
- static **Foo** * **create_data** (const **DDS_TypeAllocationParams_t** &alloc_params=DDS_TYPE_ALLOCATION↩_PARAMS_DEFAULT)

  <<***extension***>> *(p. 432) Create a data type and initialize it.*
- static **Foo** * **create_data_ex** (**DDS_Boolean** allocatePointers)

  <<***extension***>> *(p. 432) Create a data type and initialize it.*
- static **DDS_ReturnCode_t copy_data** (**Foo** *dst_data, const **Foo** *src_data)

  <<***extension***>> *(p. 432) Copy data type.*
- static **DDS_ReturnCode_t delete_data** (**Foo** *a_data, const **DDS_TypeDeallocationParams_t** &dealloc_↩params=DDS_TYPE_DEALLOCATION_PARAMS_DEFAULT)

  <<***extension***>> *(p. 432) Destroy a user data type instance.*
- static **DDS_ReturnCode_t delete_data_ex** (**Foo** *a_data, **DDS_Boolean** deletePointers)

  <<***extension***>> *(p. 432) Destroy a user data type instance.*
- static **DDS_ReturnCode_t initialize_data** (**Foo** *a_data, const **DDS_TypeAllocationParams_t** &alloc_↩params=DDS_TYPE_ALLOCATION_PARAMS_DEFAULT)

  <<***extension***>> *(p. 432) Initialize data type.*
- static **DDS_ReturnCode_t initialize_data_ex** (**Foo** *a_data, **DDS_Boolean** allocatePointers)

  <<***extension***>> *(p. 432) Initialize data type.*
- static **DDS_ReturnCode_t finalize_data** (**Foo** *a_data, const **DDS_TypeDeallocationParams_t** &dealloc↩_params=DDS_TYPE_DEALLOCATION_PARAMS_DEFAULT)

  <<***extension***>> *(p. 432) Finalize data type.*
- static **DDS_ReturnCode_t finalize_data_ex** (**Foo** *a_data, **DDS_Boolean** deletePointers)

  <<***extension***>> *(p. 432) Finalize data type.*
- static const char * **get_type_name** ()

  Get the default name for this type.
- static void **print_data** (const **Foo** *a_data)

  <<***extension***>> *(p. 432) Print value of data type to standard out.*
- static **DDS_ReturnCode_t serialize_data_to_cdr_buffer** (char *buffer, unsigned int &length, const **Foo** *a↩_data)

  <<***extension***>> *(p. 432) Serializes the input sample into a CDR buffer of octets.*
- static **DDS_ReturnCode_t serialize_data_to_cdr_buffer_ex** (char *buffer, unsigned int &length, const **Foo** *a_data, **DDS_DataRepresentationId_t** representation)

  <<***extension***>> *(p. 432) Serializes the input sample into a buffer of octets.*
- static **DDS_ReturnCode_t deserialize_data_from_cdr_buffer** (**Foo** *sample, const char *buffer, unsigned int length)

  <<***extension***>> *(p. 432) Deserializes a sample from a buffer of octets.*
- static **DDS_ReturnCode_t data_to_string** (**Foo** *sample, char *str, **DDS_UnsignedLong** &str_size, **DDS↩_PrintFormatProperty** &property)

  <<***extension***>> *(p. 432) Transforms a data sample into a human-readable string representation.*
- static **DDS_TypeCode** * **get_typecode** ()

  <<***extension***>> *(p. 432) Retrieves the TypeCode for the Type.*

9.314.1 Detailed Description

*<<**interface**>> (p. 433) <<**generic**>> (p. 433)* User data type specific interface.

Defines the user data type specific interface generated for each application class.

The concrete user data type automatically generated by the implementation is an incarnation of this class.

See also

DDS_TYPESUPPORT_CPP (p. 74)

the `Code Generator User's Manual`

9.314.2 Member Function Documentation

9.314.2.1 register_type()

```
static DDS_ReturnCode_t FooTypeSupport::register_type (
            DDSDomainParticipant * participant,
            const char * type_name ) [static]
```

Allows an application to communicate to RTI Connext the existence of a data type.

The *generated* implementation of the operation embeds all the knowledge that has to be communicated to the middleware in order to make it able to manage the contents of data of that type. This includes in particular the key definition that will allow RTI Connext to distinguish different instances of the same type.

The same **DDSTypeSupport** (p. 1897) can be registered multiple times with a **DDSDomainParticipant** (p. 1619) using the same or different values for the `type_name`. If `register_type` is called multiple times on the same **DDSType↩Support** (p. 1897) with the same **DDSDomainParticipant** (p. 1619) and `type_name`, the second (and subsequent) registrations are ignored but the operation returns **DDS_RETCODE_OK** (p. 532).

Precondition

Cannot use the same `type_name` to register two different **DDSTypeSupport** (p. 1897) with the same **DDSDomainParticipant** (p. 1619), or else the operation will fail and **DDS_RETCODE_PRECONDITION_↩NOT_MET** (p. 532) will be returned.

Parameters

participant	*<<**in**>> (p. 433)* the **DDSDomainParticipant** (p. 1619) to register the data type **Foo** (p. 1916) with. Cannot be NULL.
type_name	*<<**in**>> (p. 433)* the type name under with the data type **Foo** (p. 1916) is registered with the participant; this type name is used when creating a new **DDSTopic** (p. 1885). (See **DDSDomainParticipant::create_topic** (p. 1650).) The name may not be NULL or longer than 255 characters.

Exceptions

One	of the **Standard Return Codes** (p. 532), **DDS_RETCODE_PRECONDITION_NOT_MET** (p. 532) or **DDS_RETCODE_OUT_OF_RESOURCES** (p. 533).

MT Safety:

UNSAFE on the FIRST call. It is not safe for two threads to simultaneously make the first call to register a type. Subsequent calls are thread safe.

See also

DDSDomainParticipant::create_topic (p. 1650)

9.314.2.2 unregister_type()

```
static DDS_ReturnCode_t FooTypeSupport::unregister_type (
            DDSDomainParticipant * participant,
            const char * type_name ) [static]
```

<<***extension***>> *(p. 432)* Allows an application to unregister a data type from RTI Connext. After calling unregister_type, no further communication using that type is possible.

The *generated* implementation of the operation removes all the information about a type from RTI Connext. No further communication using that type is possible.

Precondition

A type with `type_name` is registered with the participant and all **DDSTopic** (p. 1885) objects referencing the type have been destroyed. If any **DDSTopic** (p. 1885) is associated with the type, the operation will fail with **DDS_RETCODE_ERROR** (p. 532).

Postcondition

All information about the type is removed from RTI Connext. No further communication using this type is possible.

Parameters

participant	<<***in***>> *(p. 433)* the **DDSDomainParticipant** (p. 1619) to unregister the data type **Foo** (p. 1916) from. Cannot be NULL.
type_name	<<***in***>> *(p. 433)* the type name under with the data type **Foo** (p. 1916) is registered with the participant. The name should match a name that has been previously used to register a type with the participant. Cannot be NULL.

Exceptions

One	of the **Standard Return Codes** (p. 532), **DDS_RETCODE_BAD_PARAMETER** (p. 532) or **DDS_RETCODE_ERROR** (p. 532)

MT Safety:

SAFE.

See also

FooTypeSupport::register_type (p. 1981)

9.314.2.3 create_data()

```
static  Foo * FooTypeSupport::create_data (
              const  DDS_TypeAllocationParams_t & alloc_params = DDS_TYPE_ALLOCATION_PARAMS_DEFAULT
)  [static]
```

*<<**extension**>> (p. 432)* Create a data type and initialize it.

The *generated* implementation of the operation knows how to instantiate a data type and initialize it properly.

By default all memory for the type is deeply allocated, except for optional members.

Parameters

alloc_params	*<<**in**>> (p. 433)* Whether or not to recursively allocate pointers and/or optional members

Returns

Newly created data type, or NULL on failure.

See also

FooTypeSupport::delete_data_ex (p. 1985)

9.314.2.4 create_data_ex()

```
static  Foo * FooTypeSupport::create_data_ex (
              DDS_Boolean allocatePointers )  [static]
```

<<***extension***>> *(p. 432)* Create a data type and initialize it.

The *generated* implementation of the operation knows how to instantiate a data type and initialize it properly.

When `allocatePointers` is **DDS_BOOLEAN_TRUE** (p. 513), all the references (pointers) in the type are recursively allocated.

Parameters

allocatePointers	<<***in***>> *(p. 433)* Whether or not to recursively allocate pointers.

Returns

Newly created data type, or NULL on failure.

See also

FooTypeSupport::delete_data_ex (p. 1985)

9.314.2.5 copy_data()

```
static  DDS_ReturnCode_t FooTypeSupport::copy_data (
              Foo * dst_data,
             const  Foo * src_data )  [static]
```

<<***extension***>> *(p. 432)* Copy data type.

The *generated* implementation of the operation knows how to copy value of a data type.

Parameters

dst_data	<<***inout***>> *(p. 433)* Data type to copy value to. Cannot be NULL.
src_data	<<***in***>> *(p. 433)* Data type to copy value from. Cannot be NULL.

Exceptions

One	of the **Standard Return Codes** (p. 532)

9.314.2.6 delete_data()

```
static  DDS_ReturnCode_t FooTypeSupport::delete_data (
              Foo * a_data,
```

```
            const  DDS_TypeDeallocationParams_t & dealloc_params = DDS_TYPE_DEALLOCATION_PARAMS↩
_DEFAULT )  [static]
```

<<***extension***>> (p. *432)* Destroy a user data type instance.

The *generated* implementation of the operation knows how to destroy a data type and return all resources.

Parameters

a_data	<<***in***>> (p. *433)* Cannot be NULL.
dealloc_params	<<***in***>> (p. *433)* Whether or not to destroy pointers and/or optional members. By default they are deleted.

Exceptions

One	of the **Standard Return Codes** (p. 532)

See also

FooTypeSupport::create_data (p. 1983)

9.314.2.7 delete_data_ex()

```
static  DDS_ReturnCode_t FooTypeSupport::delete_data_ex (
              Foo * a_data,
              DDS_Boolean deletePointers )  [static]
```

<<***extension***>> (p. *432)* Destroy a user data type instance.

The *generated* implementation of the operation knows how to destroy a data type and return all resources.

When `deletePointers` is **DDS_BOOLEAN_TRUE** (p. 513), all the references (pointers) are destroyed as well.

Parameters

a_data	<<***in***>> (p. *433)* Cannot be NULL.
deletePointers	<<***in***>> (p. *433)* Whether or not to destroy pointers.

Exceptions

One	of the **Standard Return Codes** (p. 532)

See also

FooTypeSupport::create_data_ex (p. 1983)

9.314.2.8 initialize_data()

```
static DDS_ReturnCode_t FooTypeSupport::initialize_data (
            Foo * a_data,
           const DDS_TypeAllocationParams_t & alloc_params = DDS_TYPE_ALLOCATION_PARAMS_DEFAULT
) [static]
```

<<***extension***>> *(p. 432)* Initialize data type.

The *generated* implementation of the operation knows how to initialize a data type. This method is typically called to initialize a data type that is allocated on the stack. Calling this method more than once will cause a memory leak.

The `alloc_params` determine whether or not all of the references (pointers) and optional members in the type are recursively allocated.

Parameters

a_data	<<***inout***>> *(p. 433)* Cannot be NULL.
alloc_params	<<***in***>> *(p. 433)* Parameters that determine whether or not to recursively allocate pointers and optional members.

Exceptions

One	of the **Standard Return Codes** (p. 532)

See also

FooTypeSupport::finalize_data (p. 1987)

9.314.2.9 initialize_data_ex()

```
static DDS_ReturnCode_t FooTypeSupport::initialize_data_ex (
            Foo * a_data,
            DDS_Boolean allocatePointers ) [static]
```

<<***extension***>> *(p. 432)* Initialize data type.

The *generated* implementation of the operation knows how to initialize a data type. This method is typically called to initialize a data type that is allocated on the stack. Calling this method more than once will cause a memory leak.

When `allocatePointers` is **DDS_BOOLEAN_TRUE** (p. 513), all the references (pointers) in the type are recursively allocated.

Parameters

a_data	<<***inout***>> *(*p. *433)* Cannot be NULL.
allocatePointers	<<***in***>> *(*p. *433)* Whether or not to recursively allocate pointers.

Exceptions

One	of the **Standard Return Codes** (p. 532)

See also

FooTypeSupport::finalize_data_ex (p. 1987)

9.314.2.10 finalize_data()

```
static DDS_ReturnCode_t FooTypeSupport::finalize_data (
              Foo * a_data,
             const DDS_TypeDeallocationParams_t & dealloc_params = DDS_TYPE_DEALLOCATION_PARAMS↩
_DEFAULT )  [static]
```

<<***extension***>> *(*p. *432)* Finalize data type.

The *generated* implementation of the operation knows how to finalize a data type. This method is typically called to finalize a data type that has previouslly been initialized.

Parameters

a_data	<<***in***>> *(*p. *433)* Cannot be NULL.
dealloc_params	<<***in***>> *(*p. *433)* Whether or not to destroy pointers and/or optional members. By default they are deleted.

Exceptions

One	of the **Standard Return Codes** (p. 532)

See also

FooTypeSupport::initialize_data (p. 1986)

9.314.2.11 finalize_data_ex()

```
static DDS_ReturnCode_t FooTypeSupport::finalize_data_ex (
            Foo * a_data,
            DDS_Boolean deletePointers ) [static]
```

<<***extension***>> *(p. 432)* Finalize data type.

The *generated* implementation of the operation knows how to finalize a data type. This method is typically called to finalize a data type that has previouslly been initialized.

When `deletePointers` is **DDS_BOOLEAN_TRUE** (p. 513), the memory required by the references (pointers) associated to the type is freed.

Parameters

a_data	<<***in***>> *(p. 433)* Cannot be NULL.
deletePointers	<<***in***>> *(p. 433)* Whether or not to free memory allocated by the pointers.

Exceptions

One	of the **Standard Return Codes** (p. 532)

See also

FooTypeSupport::initialize_data_ex (p. 1986)

9.314.2.12 get_type_name()

```
static const char * FooTypeSupport::get_type_name ( ) [static]
```

Get the default name for this type.

Can be used for calling **FooTypeSupport::register_type** (p. 1981) or creating **DDSTopic** (p. 1885)

Returns

default name for this type

See also

FooTypeSupport::register_type (p. 1981)

DDSDomainParticipant::create_topic (p. 1650)

9.314.2.13 print_data()

```
static void FooTypeSupport::print_data (
            const  Foo * a_data )  [static]
```

<<***extension***>> *(p. 432)* Print value of data type to standard out.

The *generated* implementation of the operation knows how to print value of a data type.

Parameters

a_data	<<***in***>> *(p. 433)* Data type to be printed.

9.314.2.14 serialize_data_to_cdr_buffer()

```
static  DDS_ReturnCode_t FooTypeSupport::serialize_data_to_cdr_buffer (
            char * buffer,
            unsigned int & length,
            const  Foo * a_data )  [static]
```

<<***extension***>> *(p. 432)* Serializes the input sample into a CDR buffer of octets.

This method serializes a sample into a buffer of octets using Common Data Representation (CDR). Calling this method is equivalent to calling **FooTypeSupport::serialize_data_to_cdr_buffer_ex** (p. 1990) with **DDS_AUTO_↩ DATA_REPRESENTATION** (p. 569) as the representation.

The input buffer must be big enough to store the serialized representation of the sample. Otherwise, the method will return an error.

To determine the minimum size of the input buffer, the user must call this method with the buffer set to NULL.

For more information about CDR in Connext, see `Data Representation` in the Core Libraries Extensible Types Guide.

Parameters

a_data	<<***in***>> *(p. 433)*. Input sample. Cannot be NULL.
buffer	<<***out***>> *(p. 433)*. Serialization buffer.
length	<<***inout***>> *(p. 433)*. When `buffer` is set to NULL, after the method executes, `length` will contain a buffer size big enough to hold the serialized data. When `buffer` is not NULL, `length` must contain the size of the input buffer when the method is invoked. After the method executes, `length` will be updated to contain the actual size of the serialized content, which may be smaller than the size obtained when `buffer` is set to NULL.

Exceptions

One	of the **Standard Return Codes** (p. 532)

9.314.2.15 serialize_data_to_cdr_buffer_ex()

```
static  DDS_ReturnCode_t FooTypeSupport::serialize_data_to_cdr_buffer_ex (
             char * buffer,
             unsigned int & length,
             const  Foo * a_data,
              DDS_DataRepresentationId_t representation )  [static]
```

<<***extension***>> *(p. 432)* Serializes the input sample into a buffer of octets.

This method serializes a sample into a buffer of octets using the input data representation. See **FooTypeSupport::serialize_data_to_cdr_buffer** (p. 1989) for details.

Parameters

a_data	<<***in***>> *(p. 433)*. Input sample. Cannot be NULL.
buffer	<<***out***>> *(p. 433)*. Serialization buffer.
length	<<***inout***>> *(p. 433)*. Serialization buffer length.
representation	<<***in***>> *(p. 433)*. Representation used to serialize the data.

Exceptions

One	of the **Standard Return Codes** (p. 532)

9.314.2.16 deserialize_data_from_cdr_buffer()

```
static  DDS_ReturnCode_t FooTypeSupport::deserialize_data_from_cdr_buffer (
              Foo * sample,
             const char * buffer,
             unsigned int length )  [static]
```

<<***extension***>> *(p. 432)* Deserializes a sample from a buffer of octets.

This method deserializes a sample from a CDR buffer of octets.

The content of the buffer generated by the method **FooTypeSupport::serialize_data_to_cdr_buffer** (p. 1989) can be provided to this method to get the sample back.

Parameters

sample	<<***in***>> *(p. 433)*. Output sample. Cannot be NULL.
buffer	<<***in***>> *(p. 433)*. Deserialization buffer. Cannot be NULL.
length	<<***in***>> *(p. 433)*. Length of the serialized representation of the sample in the buffer.

Exceptions

One	of the **Standard Return Codes** (p. 532)

9.314.2.17 data_to_string()

```
static DDS_ReturnCode_t FooTypeSupport::data_to_string (
            Foo * sample,
           char * str,
            DDS_UnsignedLong & str_size,
            DDS_PrintFormatProperty & property ) [static]
```

<<***extension***>> *(p. 432)* Transforms a data sample into a human-readable string representation.

This method takes a data sample and creates a string representation of the data.

The input character buffer must be big enough to store the string representation of the sample. Otherwise, the method will return an error.

To determine the minimum size of the input character buffer, the user must call this method with the buffer set to NULL.

If the size of the output string is longer than the size of an unsigned 32-bit integer, this operation will fail with **DDS_RETCODE_OUT_OF_RESOURCES** (p. 533).

This method is only provided for types that were generated with typecodes.

Parameters

sample	<<***in***>> *(p. 433)*. The sample to get the string representation for. Cannot be NULL.
str	<<***out***>> *(p. 433)*. Output string representing the data sample.
str_size	<<***inout***>> *(p. 433)*. When `str` is set to NULL, after the method executes, `str_size` will contain a buffer size big enough to hold the string representation of the data. When `str` is not NULL, `str_size` must contain the size of the input buffer when the method is invoked. If the size of the input buffer is too small, after the method executes, `str_size` will be updated to contain the required size of the string content and the method will return **DDS_RETCODE_OUT_OF_RESOURCES** (p. 533).
property	<<***in***>> *(p. 433)*. Properties describing what the format of the output string should be.

Exceptions

One	of the **Standard Return Codes** (p. 532), **DDS_RETCODE_OUT_OF_RESOURCES** (p. 533)

9.314.2.18 get_typecode()

```
static DDS_TypeCode * FooTypeSupport::get_typecode ( ) [static]
```

<<***extension***>> *(p. 432)* Retrieves the TypeCode for the Type.

This method retrieves the **DDS_TypeCode** (p. 1428) for the Type. A **DDS_TypeCode** (p. 1428) is a mechanism for representing a type at runtime. RTI Connext can use type codes to send type definitions on the network. A **DDS_TypeCode** (p. 1428) value consists of a type code *kind* (represented by the **DDS_TCKind** (p. 89) enumeration) and a list of *members* (that is, fields). These members are recursive: each one has its own **DDS_TypeCode** (p. 1428), and in the case of complex types (structures, arrays, and so on), these contained type codes contain their own members.

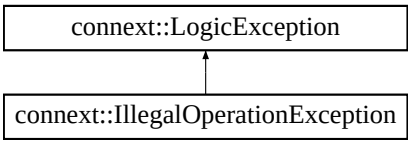
Returns

The TypeCode for this type

9.315 connext::IllegalOperationException Class Reference

The operation was called under improper circumstances.

Inheritance diagram for connext::IllegalOperationException:



9.315.1 Detailed Description

The operation was called under improper circumstances.

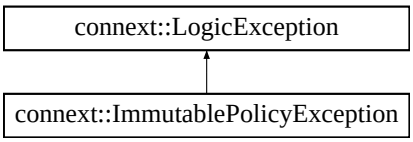
See also

DDS_RETCODE_ILLEGAL_OPERATION (p. 533)

9.316 connext::ImmutablePolicyException Class Reference

Application attempted to modify an immutable QoS policy.

Inheritance diagram for connext::ImmutablePolicyException:



9.316.1 Detailed Description

Application attempted to modify an immutable QoS policy.

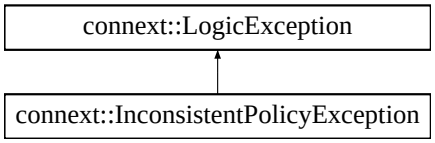
See also

DDS_RETCODE_IMMUTABLE_POLICY (p. 533)

9.317 connext::InconsistentPolicyException Class Reference

Application specified a set of QoS policies that are not consistent with each other.

Inheritance diagram for connext::InconsistentPolicyException:



9.317.1 Detailed Description

Application specified a set of QoS policies that are not consistent with each other.

See also

DDS_RETCODE_INCONSISTENT_POLICY (p. 533)

9.318 connext::IsInvalidSamplePredicate< T > Class Template Reference

Predicate-class to determine if a sample contains invalid data.

9.318.1 Detailed Description

template<typename T>
class connext::IsInvalidSamplePredicate< T >

Predicate-class to determine if a sample contains invalid data.

9.319 connext::IsReplyRelatedPredicate< T > Class Template Reference

Predicate-class to match replies by their related request.

Public Member Functions

- **IsReplyRelatedPredicate** (const **DDS::SampleIdentity_t** &related_request_id)

  Creates the predicate with the request id to match.
- bool **operator()** (**SampleRef**< T > sample)

  Determines if a reply is related to the request id this object contains.

9.319.1 Detailed Description

template<typename T>
class connext::IsReplyRelatedPredicate< T >

Predicate-class to match replies by their related request.

Useful for C++ standard algorithms like `std::find_if()`

See also

Correlating requests and replies (p. 423)

9.319.2 Constructor & Destructor Documentation

9.319.2.1 IsReplyRelatedPredicate()

```
template<typename T >
connext::IsReplyRelatedPredicate< T >::IsReplyRelatedPredicate (
            const  DDS::SampleIdentity_t & related_request_id )  [inline]
```

Creates the predicate with the request id to match.

9.319.3 Member Function Documentation

9.319.3.1 operator()()

```
template<typename T >
bool  connext::IsReplyRelatedPredicate< T >::operator() (
             SampleRef< T > sample )  [inline]
```

Determines if a reply is related to the request id this object contains.

9.320 connext::IsValidSamplePredicate< T > Class Template Reference

Predicate-class to determine if a sample contains valid data.

9.320.1 Detailed Description

template<typename T>
class connext::IsValidSamplePredicate< T >

Predicate-class to determine if a sample contains valid data.

Useful for C++ standard algorithms like `std::copy()`

See also

Taking samples by copy (p. 422)

9.321 connext::LoanedSamples< T > Class Template Reference

Provides access to a collection of middleware-loaned samples.

Classes

- struct **LoanMemento**

  *A simple value-type for the internal representation of the a **LoanedSamples** (p. 1995) object.*

Public Types

- typedef **SampleIterator**< T, false > **iterator**

  The iterator type.
- typedef **SampleIterator**< T, true > **const_iterator**

  The const iterator type.

Public Member Functions

- **LoanedSamples** ()

  *Creates an empty **LoanedSamples** (p. 1995) object.*
- void **release** (TDataReader *&reader_ptr, TSeq &data_seq, **SampleInfoSeq** &info_seq)

  *Transfers the ownership of the underlying data and the **DDS_SampleInfo** (p. 1344) sequences.*
- **LoanedSamples** (**LoanMemento** loan_memento) throw ()

  *Reconstruct a new **LoanedSamples** (p. 1995) object from an internal representation of another.*
- ~**LoanedSamples** () throw ()

  Automatically returns the loan to the middleware.
- **value_type operator[ ]** (size_t index)

  *Provides access to the underlying **SampleRef** (p. 2191) object in array-like syntax.*
- **const_value_type operator[ ]** (size_t index) const

  *Provides access to the underlying **SampleRef** (p. 2191) object in array-like syntax.*
- int **length** () const

  Returns the number of samples.
- void **return_loan** ()

  Returns the loaned samples to the middleware.
- **operator LoanMemento** () throw ()

  Release the ownership of the underlying loaned data and SampleInfo and returns an internal representation object.
- **iterator begin** ()

  Provides an iterator to the first element in the container.
- **iterator end** ()

  Provides an iterator to the past-the-end element in the container.
- **const_iterator begin** () const

  Provides an iterator to the first element in the container.
- **const_iterator end** () const

  Provides an iterator to the past-the-end element in the container.

Static Public Member Functions

- static **LoanedSamples move_construct_from_loans** (TDataReader *reader, TSeq &data_seq, **SampleInfoSeq** &info_seq)

  *Create a new **LoanedSamples** (p. 1995) object by moving the ownership of the data sequence and the **DDS_SampleInfo** (p. 1344) sequences.*

9.321.1 Detailed Description

template<typename T>
class connext::LoanedSamples< T >

Provides access to a collection of middleware-loaned samples.

The samples in this container are loaned from the middleware and must be returned at some point.

The loan is automatically returned in the destructor. Alternatively **return_loan** (p. 2002) can be used.

This is a move-only type. That is, The copy-construction and copy-assignment operations are explicitly disabled. Inspite of that the objects of **LoanedSamples** (p. 1995) type can be passed to a function and returned from a function by-value very efficiently. The **connext::details::move()** (p. 195) should be used while passing **LoanedSamples** (p. 1995) by value in and out of a function. Moreover, as assignments are disabled, **connext::details::move()** (p. 195) must be used to transfer the ownership of the loaned samples from one object to another. **connext::details::move()** (p. 195) ensures that there exists exactly one owner of the loan.

Due to its move-only nature, instances of **LoanedSamples** (p. 1995) can't be stored in C++ standard library containers. There are two possible ways to achieve this. Both are not exception safe.

First, use dynamically allocated LoanedSamples<T> objects and the STL containers would store the pointers to the LoanedSamples<T> objects. The **LoanedSamples** (p. 1995) objects returned by the middlware can be moved in to the dynamically allocated LoanedSamples<T> objects using **connext::details::move()** (p. 195) . Using smart pointers for automatic memory management is highly recommened if this approach is chosen.

Second alternative is to use the **LoanMemento** (p. 2004) conversion operator. An STL container of LoanedSamples<↩ T>::LoanMemento objects can be created from a LoanedSamples<T> object. LoanedSamples<T> object releases it ownership of the resources in this conversion. Therefore, when the **LoanMemento** (p. 2004) object is no longer required, the underlying resources must be returned by first creating a **LoanedSamples** (p. 1995) object from the **LoanMemento** (p. 2004). This technique does not require a dynamic allocation like the first alternative. However, this technique should be used very carefully because in case of exceptions, **LoanMemento** (p. 2004) object will not free the resources like the **LoanedSamples** (p. 1995) object does.

This container provides STL-compliant random-access iterators (**begin** (p. 2003) and **end** (p. 2003)).

The contained elements are of type **connext::SampleRef** (p. 2191) Therfore, dereferencing a **LoanedSamples** (p. 1995) iterator returns a **connext::SampleRef** (p. 2191) object, which in turn refer to the data and **DDS_SampleInfo** (p. 1344).

The contents of this container should not be modified and references to the samples it contains are only valid before the loan is returned.

Template Parameters

T	The data type of the contained Samples

See also

connext::Requester::take_replies(int) (p. 2170)

connext::Replier::take_requests(int) (p. 2149)

Taking loaned samples (p. 422)

Taking samples by copy (p. 422)

9.321.2 Member Typedef Documentation

9.321.2.1 iterator

```
template<typename T >
typedef  SampleIterator<T, false>  connext::LoanedSamples< T >::iterator
```

The iterator type.

9.321.2.2 const_iterator

```
template<typename T >
typedef  SampleIterator<T, true>  connext::LoanedSamples< T >::const_iterator
```

The const iterator type.

9.321.3 Constructor & Destructor Documentation

9.321.3.1 LoanedSamples() [1/2]

```
template<typename T >
connext::LoanedSamples< T >::LoanedSamples ( )  [inline]
```

Creates an empty **LoanedSamples** (p. 1995) object.

Referenced by **connext::LoanedSamples< T >::move_construct_from_loans()**.

9.321.3.2 LoanedSamples() [2/2]

```
template<typename T >
connext::LoanedSamples< T >::LoanedSamples (
              LoanMemento loan_memento ) throw ( )   [inline]
```

Reconstruct a new **LoanedSamples** (p. 1995) object from an internal representation of another.

```
This constructor will regain the ownership of the underlying data and ::DDS_SampleInfo
sequences.
```

Parameters

loan_memento	The internal representation of another **LoanedSamples** (p. 1995) object.

See also

connext::LoanedSamples (p. 1995)

9.321.3.3 ~LoanedSamples()

```
template<typename T >
connext::LoanedSamples< T >::~ LoanedSamples ( ) throw ( )   [inline]
```

Automatically returns the loan to the middleware.

See also

return_loan (p. 2002)

References **connext::LoanedSamples< T >::return_loan()**.

9.321.4 Member Function Documentation

9.321.4.1 move_construct_from_loans()

```
template<typename T >
static  LoanedSamples  connext::LoanedSamples< T >::move_construct_from_loans (
             TDataReader * reader,
             TSeq & data_seq,
              SampleInfoSeq & info_seq )  [inline], [static]
```

Create a new **LoanedSamples** (p. 1995) object by moving the ownership of the data sequence and the **DDS_SampleInfo** (p. 1344) sequences.

Precondition

The `data_seq` and `info_seq` parameters must contained loaned sample from the same DataReader.

Parameters

reader	The DataReader from which the loan was taken.
data_seq	The **FooSeq** (p. 1965) object containing the loaned samples. After the call this sequence will be empty.
info_seq	The **DDS_SampleInfoSeq** (p. 1355) object containing the loaned SampleInfo. After this call the sequence will be empty.

Postcondition

The new **LoanedSamples** (p. 1995) object will manage the ownership of the loans.

Returns

void

See also

connext::LoanedSamples (p. 1995)

References **DDS_RETCODE_BAD_PARAMETER**, and **connext::LoanedSamples< T >::LoanedSamples()**.

9.321.4.2 release()

```
template<typename T >
void  connext::LoanedSamples< T >::release (
            TDataReader *& reader_ptr,
            TSeq & data_seq,
             SampleInfoSeq & info_seq )  [inline]
```

Transfers the ownership of the underlying data and the **DDS_SampleInfo** (p. 1344) sequences.

Precondition

The `data_seq` and `info_seq` parameters must be default initialized prior to this call.

Parameters

reader_ptr	The DataReader from which the loan was taken. After the call the pointer will be updated.
data_seq	The **FooSeq** (p. 1965) object that will contain the loan previously held by the **LoanedSamples** (p. 1995) object. After the call this sequence will loan the underlying data.
info_seq	The **DDS_SampleInfoSeq** (p. 1355) object that will contain the SampleInfo loan previously held by the **LoanedSamples** (p. 1995) object. After this call this sequence will loan the underlying **DDS_SampleInfoSeq** (p. 1355).

Postcondition

The `data_seq` and `info_seq` objects will contain the loan previously held by the **LoanedSamples** (p. 1995). The **LoanedSamples** (p. 1995) object will release its ownership.

See also

connext::LoanedSamples (p. 1995)

References **DDS_RETCODE_PRECONDITION_NOT_MET**.

9.321.4.3 operator[]() [1/2]

```
template<typename T >
value_type  connext::LoanedSamples< T >::operator[] (
             size_t index )  [inline]
```

Provides access to the underlying **SampleRef** (p. 2191) object in array-like syntax.

Parameters

index	The index of the **Sample** (p. 2183). Allowed values are from 0 to **length()** (p. 2002)-1.

Returns

A **connext::SampleRef** (p. 2191) object that refers to data and **DDS_SampleInfo** (p. 1344) at the specified `index`.

See also

connext::LoanedSamples (p. 1995)

9.321.4.4 operator[]() [2/2]

```
template<typename T >
const_value_type  connext::LoanedSamples< T >::operator[] (
             size_t index ) const  [inline]
```

Provides access to the underlying **SampleRef** (p. 2191) object in array-like syntax.

Parameters

index	The index of the **Sample** (p. 2183). Allowed values are from 0 to **length()** (p. 2002)-1.

Returns

A **connext::SampleRef** (p. 2191) object that refers to a constant data and constant **DDS_SampleInfo** (p. 1344) at the specified `index`.

See also

connext::LoanedSamples (p. 1995)

9.321.4.5 length()

```
template<typename T >
int  connext::LoanedSamples< T >::length ( ) const  [inline]
```

Returns the number of samples.

9.321.4.6 return_loan()

```
template<typename T >
void  connext::LoanedSamples< T >::return_loan ( )  [inline]
```

Returns the loaned samples to the middleware.

After calling this operation this object cannot be accessed again.

Calling this operation is optional, since the destructor will automatically do it.

See also

FooDataReader::return_loan (p. 1940) (for more information on how the middleware loans samples)

Referenced by **connext::LoanedSamples< T >::~LoanedSamples()**.

9.321.4.7 operator LoanMemento()

```
template<typename T >
connext::LoanedSamples< T >::operator  LoanMemento ( ) throw ( )   [inline]
```

Release the ownership of the underlying loaned data and SampleInfo and returns an internal representation object.

LoanMemento (p. 2004) object has trivial copy semantics. Therefore, it does not manage resources automatically like **LoanedSamples** (p. 1995) does. This conversion function should be used very carefully. This conversion can be used to store **LoanedSamples** (p. 1995) in a standard library containers. However, note that resource management must be handled manually.

See also

connext::LoanedSamples (p. 1995)

9.321.4.8 begin() [1/2]

```
template<typename T >
iterator  connext::LoanedSamples< T >::begin ( )  [inline]
```

Provides an iterator to the first element in the container.

Returns

A STL-compliant random-access iterator to the first element in this container

See also

connext::SampleIterator (p. 2191)

end (p. 2003)

9.321.4.9 end() [1/2]

```
template<typename T >
iterator  connext::LoanedSamples< T >::end ( )  [inline]
```

Provides an iterator to the past-the-end element in the container.

Returns

A STL-compliant random-access iterator indicating the end of the container.

See also

connext::SampleIterator (p. 2191)

begin (p. 2003)

9.321.4.10 begin() [2/2]

```
template<typename T >
const_iterator connext::LoanedSamples< T >::begin ( ) const [inline]
```

Provides an iterator to the first element in the container.

Returns

A STL-compliant random-access iterator to the first element in this container

See also

connext::SampleIterator (p. 2191)
end (p. 2003)

9.321.4.11 end() [2/2]

```
template<typename T >
const_iterator connext::LoanedSamples< T >::end ( ) const [inline]
```

Provides an iterator to the past-the-end element in the container.

Returns

A STL-compliant random-access iterator indicating the end of the container.

See also

connext::SampleIterator (p. 2191)
begin (p. 2003)

9.322 connext::LoanedSamples< T >::LoanMemento Struct Reference

A simple value-type for the internal representation of the a **LoanedSamples** (p. 1995) object.

9.322.1 Detailed Description

template<typename T>
struct connext::LoanedSamples< T >::LoanMemento

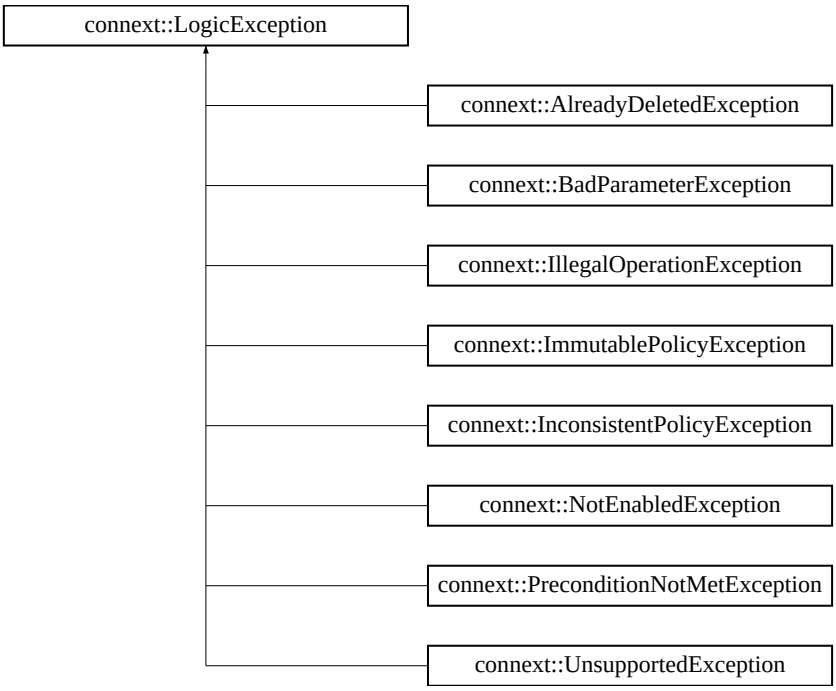
A simple value-type for the internal representation of the a **LoanedSamples** (p. 1995) object.

Objects of this type can be used to store **LoanedSamples** (p. 1995) in an STL container. Care be taken in case of exceptions because **LoanMemento** (p. 2004) does not offer support for exception-safe programming.

9.323 connext::LogicException Class Reference

Base class of all RTI Connext exceptions caused by a logic error.

Inheritance diagram for connext::LogicException:



9.323.1 Detailed Description

Base class of all RTI Connext exceptions caused by a logic error.

Its base class is `std::logic_error`, from which it inherits the `what()` method.

9.324 connext::MessagingLibraryVersion Class Reference

The Connext Messaging Library version.

Public Member Functions

- **DDS_Char major_version** () const
  Get the library major version digit.
- **DDS_Char minor_version** () const
  Get the library minor version digit.
- **DDS_Char release_version** () const
  Get the library release version digit.
- **DDS_Char build_version** () const
  Get the library revision version digit.

9.324.1 Detailed Description

The Connext Messaging Library version.

This class represents the Connext Messaging library version. The version format is <major>.<minor>.<release>.<revision>.

9.324.2 Member Function Documentation

9.324.2.1 major_version()

```
DDS_Char connext::MessagingLibraryVersion::major_version ( ) const  [inline]
```

Get the library major version digit.

9.324.2.2 minor_version()

```
DDS_Char connext::MessagingLibraryVersion::minor_version ( ) const  [inline]
```

Get the library minor version digit.

9.324.2.3 release_version()

```
DDS_Char connext::MessagingLibraryVersion::release_version ( ) const  [inline]
```

Get the library release version digit.

9.324.2.4 build_version()

```
DDS_Char connext::MessagingLibraryVersion::build_version ( ) const  [inline]
```

Get the library revision version digit.

9.325 connext::MessagingVersion Class Reference

The Connext Messaging version.

Static Public Member Functions

- static const **MessagingLibraryVersion** & **get_api_version** ()

  Get the API version.
- static const std::string & **get_api_version_string** ()

  Get the API version as a string.
- static const std::string & **get_api_build_version** ()

  Get the build version as a string.

9.325.1 Detailed Description

The Connext Messaging version.

9.325.2 Member Function Documentation

9.325.2.1 get_api_version()

```
static const  MessagingLibraryVersion & connext::MessagingVersion::get_api_version ( )  [static]
```

Get the API version.

9.325.2.2 get_api_version_string()

```
static const std::string & connext::MessagingVersion::get_api_version_string ( )  [static]
```

Get the API version as a string.

9.325.2.3 get_api_build_version()

```
static const std::string & connext::MessagingVersion::get_api_build_version ( )  [static]
```

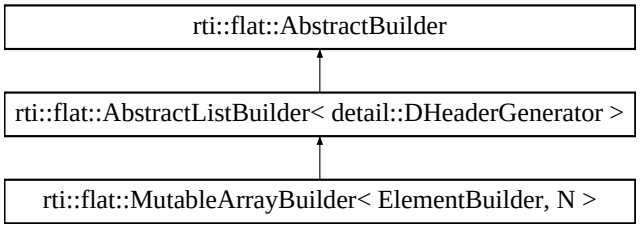
Get the build version as a string.

9.326 rti::flat::MutableArrayBuilder< ElementBuilder, N > Class Template Reference

Builds an array member of variable-size elements.

`#include <SequenceBuilders.hpp>`

Inheritance diagram for rti::flat::MutableArrayBuilder< ElementBuilder, N >:



Public Types

- typedef **MutableArrayOffset**< typename ElementBuilder::Offset, N > **Offset**

  The related Offset type.

Public Member Functions

- ElementBuilder **build_next** ()

  Begins building the next element.
- **Offset finish** ()

  Finishes building the array.

Additional Inherited Members

9.326.1 Detailed Description

template<typename ElementBuilder, unsigned int N>
class rti::flat::MutableArrayBuilder< ElementBuilder, N >

Builds an array member of variable-size elements.

Template Parameters

ElementBuilder	The Builder type for the elements of the array
N	The array bound; the exact number of elements this array builder must build. For multidimensional arrays, N is the product of each dimension bound.

Each element of this array needs to be built using the ElementBuilder returned by **build_next()** (p. 2009). N elements exactly must be built. Unlike a sequence Builder, it is not possible to finish an array with less than N elements.

This example shows how to use a **MutableArrayBuilder** (p. 2008) to build an array member of **MyFlatMutableBuilder** (p. 2018):

```
MyFlatMutableBuilder builder = rti::flat::build_data(writer);
auto array_builder = builder.build_my_mutable_array();
for (int i = 0; i < 10; i++) {
    FlatMutableBarBuilder element_builder = array_builder.build_next();
    // ... build element
    element_builder.finish();
}
array_builder.finish();
```

Note that Builder types are not necessary for arrays of fixed-size elements, since they are added at once (see **MyFlat↩ MutableBuilder::add_my_final_array()** (p. 2023)).

9.326.2 Member Typedef Documentation

9.326.2.1 Offset

```
template<typename ElementBuilder , unsigned int N>
typedef MutableArrayOffset<typename ElementBuilder::Offset, N> rti::flat::MutableArrayBuilder<
ElementBuilder, N >::Offset
```

The related Offset type.

9.326.3 Member Function Documentation

9.326.3.1 build_next()

```
template<typename ElementBuilder , unsigned int N>
ElementBuilder rti::flat::MutableArrayBuilder< ElementBuilder, N >::build_next ( ) [inline]
```

Begins building the next element.

Before calling **build_next()** (p. 2009) to create a new element, the element Builder returned by a previous call to build↩ _next must have been finished.

References **rti::flat::AbstractListBuilder< detail::DHeaderGenerator >::element_count()**.

9.326.3.2 finish()

```
template<typename ElementBuilder , unsigned int N>
Offset rti::flat::MutableArrayBuilder< ElementBuilder, N >::finish ( ) [inline]
```

Finishes building the array.

Precondition

build_next() (p. 2009) must have been succesfully called N times exactly.

Returns

The Offset to the member that has been built.

See also

discard() (p. 788)

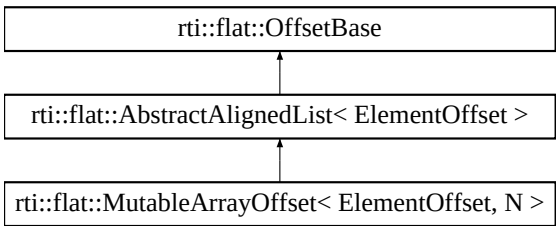
References **rti::flat::AbstractListBuilder< detail::DHeaderGenerator >::element_count()**.

9.327 rti::flat::MutableArrayOffset< ElementOffset, N > Class Template Reference

Offset to an array of variable-size elements.

`#include <SequenceOffsets.hpp>`

Inheritance diagram for rti::flat::MutableArrayOffset< ElementOffset, N >:



Public Member Functions

- ElementOffset **get_element** (unsigned int i)
  Gets the Offset to an element.

Additional Inherited Members

9.327.1 Detailed Description

template<typename ElementOffset, unsigned int N>
class rti::flat::MutableArrayOffset< ElementOffset, N >

Offset to an array of variable-size elements.

Template Parameters

ElementOffset	An Offset for a type of variable size, such as a mutable struct (**MyFlatMutableOffset** (p. 2025)), union, or **StringOffset** (p. 2216).
N	The array bound. For multidimensional arrays, N is the product of each dimension bound.

Represents an Offset to an array member and allows getting an Offset to each of its elements.

See also

FinalArrayOffset (p. 1911) encapsulates arrays of fixed-size elements

9.327.2 Member Function Documentation

9.327.2.1 get_element()

```
template<typename ElementOffset , unsigned int N>
ElementOffset  rti::flat::MutableArrayOffset< ElementOffset, N >::get_element (
             unsigned int i )  [inline]
```

Gets the Offset to an element.

Parameters

i	The zero-based index to the element

Returns

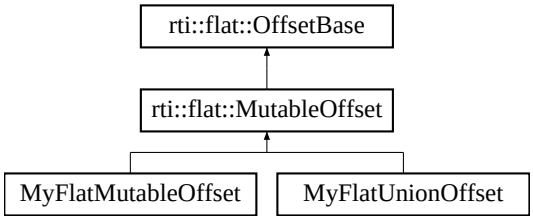
The Offset to the element in the i-th position

9.328 rti::flat::MutableOffset Class Reference

The base class of all Offsets to a final struct type.

`#include <Offset.hpp>`

Inheritance diagram for rti::flat::MutableOffset:



Additional Inherited Members

9.328.1 Detailed Description

The base class of all Offsets to a final struct type.

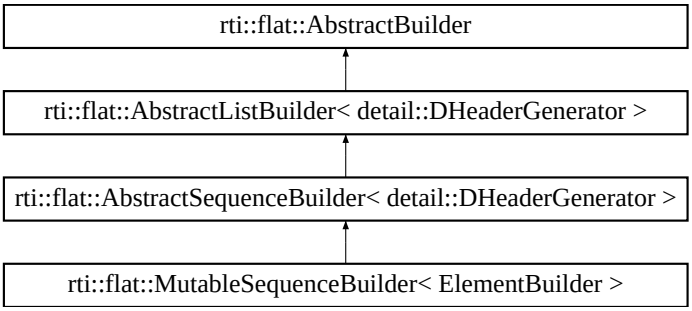
This class contains only implementation details; all the public accessors are defined in the generated type (**MyFlat↩ MutableOffset** (p. 2025)).

9.329 rti::flat::MutableSequenceBuilder< ElementBuilder > Class Template Reference

Builds a sequence member of variable-size elements.

`#include <SequenceBuilders.hpp>`

Inheritance diagram for rti::flat::MutableSequenceBuilder< ElementBuilder >:



Public Types

- typedef **SequenceOffset**< typename ElementBuilder::Offset > **Offset**
  The related Offset type.

Public Member Functions

- ElementBuilder **build_next** ()
  Begins building the next element.
- **Offset finish** ()
  Finishes building the sequence.

Additional Inherited Members

9.329.1 Detailed Description

template<typename ElementBuilder>
class rti::flat::MutableSequenceBuilder< ElementBuilder >

Builds a sequence member of variable-size elements.

Template Parameters

ElementBuilder	The Builder type for the elements of the sequence

To build the elements use the ElementBuilder returned by **build_next()** (p. 2013). An empty sequence can be built by calling **finish()** (p. 2013) without any call to **build_next()** (p. 2013).

This class doesn't enforce the sequence bound set in IDL.

The following example uses a **MutableSequenceBuilder** (p. 2012) to initialize a sequence member of **MyFlatMutable↩ Builder** (p. 2018) with two elements:

```
MyFlatMutableBuilder builder = rti::flat::build_data(writer);
auto seq_builder = builder.build_my_mutable_seq();
auto element_builder = seq_builder.build_next();
// ... build the first element
element_builder.finish();
element_builder = seq_builder.build_next();
// ... build the second element
element_builder.finish();

seq_builder.finish();
```

9.329.2 Member Typedef Documentation

9.329.2.1 Offset

```
template<typename ElementBuilder >
typedef SequenceOffset<typename ElementBuilder::Offset> rti::flat::MutableSequenceBuilder<
ElementBuilder >::Offset
```

The related Offset type.

9.329.3 Member Function Documentation

9.329.3.1 build_next()

```
template<typename ElementBuilder >
ElementBuilder rti::flat::MutableSequenceBuilder< ElementBuilder >::build_next ( ) [inline]
```

Begins building the next element.

Before calling **build_next()** (p. 2013) to create a new element, the application must have called **finish()** (p. 2013) on the previous element Builder.

9.329.3.2 finish()

```
template<typename ElementBuilder >
Offset rti::flat::MutableSequenceBuilder< ElementBuilder >::finish ( ) [inline]
```

Finishes building the sequence.

Returns

An Offset to the member that has been built.

See also

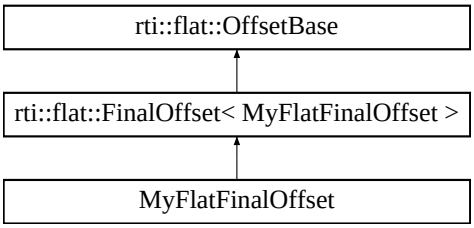
discard() (p. 788)

9.330 MyFlatFinalOffset Class Reference

Represents the Offset to an arbitrary user-defined FlatData final IDL struct.

```
#include <Offset.hpp>
```

Inheritance diagram for MyFlatFinalOffset:



Public Types

- typedef MyFlatFinalConstOffset **ConstOffset**

  The equivalent read-only Offset type.

Public Member Functions

- **MyFlatFinalOffset** ()

  Creates a null Offset.
- int32_t **my_primitive** () const

  Retrieves the value for a primitive member.
- FlatFinalBar::ConstOffset **my_complex** () const

  Retrieves a const Offset to a complex member.
- const **rti::flat::PrimitiveArrayOffset**< int32_t, 10 > **my_primitive_array** () const

  Retrieves a const Offset to a primitive array.
- **rti::flat::FinalArrayOffset**< FlatFinalBar::ConstOffset, 10 > **my_complex_array** () const

  Retrieves a const Offset to a complex array.
- bool **my_primitive** (int32_t value)

  Sets the value for a primitive member.
- FlatFinalBar::Offset **my_complex** ()

  Retrieves a non-const Offset to a complex member.
- **rti::flat::PrimitiveArrayOffset**< int32_t, 10 > **my_primitive_array** ()

  Retrieves a non-const Offset to a primitive array.
- **rti::flat::FinalArrayOffset**< FlatFinalBar::Offset, 10 > **my_complex_array** ()

  Retrieves a non-const Offset to a complex array.

9.330.1 Detailed Description

Represents the Offset to an arbitrary user-defined FlatData final IDL struct.

This example type represents the Offset type that **rtiddsgen** would generate for **MyFlatFinal** (p. 770).

It provides accessors for each of its members. Accessors can return other Offsets or primitive values.

An Offset to a final type may meet the requirements to be cast to its equivalent plain C++ type (see **rti::flat::plain_cast()** (p. 773)).

9.330.2 Member Typedef Documentation

9.330.2.1 ConstOffset

```
typedef MyFlatFinalConstOffset MyFlatFinalOffset::ConstOffset
```

The equivalent read-only Offset type.

Each Offset for a user type has an equivalent const Offset that doesn't provide the methods to modify the Sample.

9.330.3 Constructor & Destructor Documentation

9.330.3.1 MyFlatFinalOffset()

```
MyFlatFinalOffset::MyFlatFinalOffset ( )  [inline]
```

Creates a null Offset.

Postcondition

is_null() (p. 2128)

9.330.4 Member Function Documentation

9.330.4.1 my_primitive() [1/2]

```
int32_t MyFlatFinalOffset::my_primitive ( ) const
```

Retrieves the value for a primitive member.

9.330.4.2 my_complex() [1/2]

```
FlatFinalBar::ConstOffset MyFlatFinalOffset::my_complex ( ) const
```

Retrieves a const Offset to a complex member.

FlatFinalBar is another arbitrary user-defined final FlatData type.

9.330.4.3 my_primitive_array() [1/2]

```
const  rti::flat::PrimitiveArrayOffset< int32_t, 10 > MyFlatFinalOffset::my_primitive_array ( )
const
```

Retrieves a const Offset to a primitive array.

9.330.4.4 my_complex_array() [1/2]

```
rti::flat::FinalArrayOffset< FlatFinalBar::ConstOffset, 10 > MyFlatFinalOffset::my_complex_array
( ) const
```

Retrieves a const Offset to a complex array.

9.330.4.5 my_primitive() [2/2]

```
bool MyFlatFinalOffset::my_primitive (
            int32_t value )
```

Sets the value for a primitive member.

9.330.4.6 my_complex() [2/2]

```
FlatFinalBar::Offset MyFlatFinalOffset::my_complex ( )
```

Retrieves a non-const Offset to a complex member.

FlatFinalBar is another arbitrary user-defined final FlatData type.

9.330.4.7 my_primitive_array() [2/2]

```
rti::flat::PrimitiveArrayOffset< int32_t, 10 > MyFlatFinalOffset::my_primitive_array ( )
```

Retrieves a non-const Offset to a primitive array.

9.330.4.8 my_complex_array() [2/2]

```
rti::flat::FinalArrayOffset< FlatFinalBar::Offset, 10 > MyFlatFinalOffset::my_complex_array ( )
```

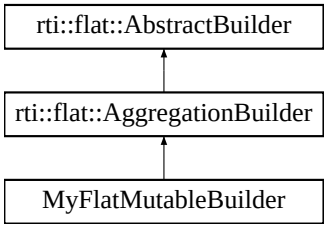
Retrieves a non-const Offset to a complex array.

9.331 MyFlatMutableBuilder Class Reference

Represents the Builder for an arbitrary user-defined mutable type.

`#include <Builder.hpp>`

Inheritance diagram for MyFlatMutableBuilder:



Public Types

- typedef **MyFlatMutableOffset Offset**
  The related offset type.

Public Member Functions

- **MyFlatMutableBuilder** ()
  Creates an invalid Builder.
- **MyFlatMutableBuilder** (unsigned char ∗buffer, int32_t size, bool initialize_members=false)
  Construct a Builder with an arbitrary buffer.
- **Offset finish** ()
  Finishes building a member.
- **MyFlatMutable** ∗ **finish_sample** ()
  Finishes building a sample.
- bool **add_my_primitive** (int32_t value)
  Adds a primitive member.
- bool **add_my_optional_primitive** (int32_t value)
  Adds a primitive member.
- **rti::flat::PrimitiveArrayOffset**< int32_t, 10 > **add_my_primitive_array** ()
  Adds a primitive array member.
- **rti::flat::PrimitiveSequenceBuilder**< int32_t > **build_my_primitive_seq** ()
  Begins building a primitive-sequence member.
- **MyFlatFinalOffset add_my_final** ()
  Adds a final complex member.
- **rti::flat::FinalAlignedArrayOffset**< **MyFlatFinalOffset**, 10 > **add_my_final_array** ()
  Adds an array member of final complex elements.
- **rti::flat::FinalSequenceBuilder**< **MyFlatFinalOffset** > **build_my_final_seq** ()
  Begins building a sequence member of final complex elements.
- FlatMutableBarBuilder **build_my_mutable** ()

Begins building a mutable complex member.

- **rti::flat::MutableArrayBuilder**< FlatMutableBarBuilder, 10 > **build_my_mutable_array** ()

  Begins building an array member of mutable complex elements.
- **rti::flat::MutableSequenceBuilder**< FlatMutableBarBuilder > **build_my_mutable_seq** ()

  Begins building a sequence member of mutable complex elements.
- **rti::flat::StringBuilder build_my_string** ()

  Begins building a string member.
- **rti::flat::MutableSequenceBuilder**< **rti::flat::StringBuilder** > **build_my_string_seq** ()

  Begins building a sequence member of string elements.

Additional Inherited Members

9.331.1 Detailed Description

Represents the Builder for an arbitrary user-defined mutable type.

This example type represents the **Builder** (p. 766) type that **rtiddsgen** would generate for **MyFlatMutable** (p. 770) and allows creating a sample or a member of that type.

The most common way to create a Builder is **rti::flat::build_data()** (p. 768), which obtains a managed sample from a **FooDataWriter** (p. 1944) as described in **Publishing FlatData** (p. 777). It is also possible to create a Builder with an arbitrary buffer using the constructor that receives the buffer and its size.

When a Builder is created, the type is empty–it doesn't contain any members. The Builder provides functions to create each member. There are two kind of functions: `add` functions and `build` functions.

9.331.1.1 Adding fixed-size members

Fixed-size members are "added", and the corresponding function is called **`add_<member_name>`**. "Add" functions directly place the member in the buffer, and return an **Offset** (p. 771) that allows setting its values.

By default, after calling `add_<member_name>` the member values are uninitialized. This behavior can be changed by creating the writer with **DDS_DataWriterResourceLimitsQosPolicy::initialize_writer_loaned_sample** (p. 925); this is however not recommended in general because of the performance impact for large types.

```
MyFlatMutableBuilder builder = ...;
MyFinalFooOffset member_offset = builder.add_my_final();
member_offset.my_primitive(10);
// ... add/build more members
```

9.331.1.2 Building variable-size members

Variable-size members are "built", and the function is called **`build_<member_name>`**. "Build" functions return another **Builder** (p. 766) to build the member.

```
MyFlatMutableBuilder builder = ...;
auto member_builder = builder.build_my_mutable();
// ... use member_builder
member_builder.finish();
// ... add/build more members
```

While a member is being built, the parent Builder is a **bound** state, which prevents any changes to it until the member has been finished by calling `member_builder.finish()`. In particular, the member Builder must be finished before adding or building any other member, or before finishing `builder`.

9.331.1.3 Choosing which members are included

The `add/build` function for each member can be called one or zero times. That is, all members are optional, even those without the `@optional` IDL annotation. However `@key` member must be added or **FooDataWriter::write** (p. 1951) will fail.

FlatData samples received by **DDSDataReader** (p. 1556) will not contain the members that were not added/built–the corresponding member getters return a null Offset. A **DDSDataReader** (p. 1556) for an equivalent non-FlatData (plain) definition of this type will assign default values to any non-optional members that were not present in the sample when written.

It is not permitted to add a member more than once, but builders don't enforce this. Trying to build/add a member more than once may cause the Builder to overflow (see **Builder Error Management** (p. 767)). If it doesn't, it may be possible to write and read the data samples, but the duplicate members will be ignored.

See also

FlatData Builders (p. 766)

9.331.2 Member Typedef Documentation

9.331.2.1 Offset

`typedef` **MyFlatMutableOffset MyFlatMutableBuilder::Offset**

The related offset type.

9.331.3 Constructor & Destructor Documentation

9.331.3.1 MyFlatMutableBuilder() [1/2]

`MyFlatMutableBuilder::MyFlatMutableBuilder ( ) [inline]`

Creates an invalid Builder.

Postcondition

!is_valid()

Note

Top-level builders are created with **rti::flat::build_data()** (p. 768), and member builders are created with the corresponding `build_<member>` function.

9.331.3.2 MyFlatMutableBuilder() [2/2]

```
MyFlatMutableBuilder::MyFlatMutableBuilder (
            unsigned char * buffer,
            int32_t size,
            bool initialize_members = false )  [inline]
```

Construct a Builder with an arbitrary buffer.

Note

The recommended way to create a Builder is **rti::flat::build_data()** (p. 768) as described in **Publishing FlatData** (p. 777). This constructor provides an alternative option to use an arbitrary data buffer.

Parameters

buffer	The buffer that will be used to build the data sample
size	The size of the buffer
initialize_members	Whether fixed-size elements added to this Builder will be initialized to their default values or will be left uninitialized (more efficient).

If the sample being built overflows the buffer size, the add/build operations will fail. See **Builder Error Management** (p. 767).

9.331.4 Member Function Documentation

9.331.4.1 finish()

```
Offset MyFlatMutableBuilder::finish ( )
```

Finishes building a member.

Returns

The Offset to the member (normally it can be ignored)

Precondition

is_nested() (p. 788). That is, this object must be a member Builder, not a sample Builder.

9.331.4.2 finish_sample()

```
MyFlatMutable * MyFlatMutableBuilder::finish_sample ( )
```

Finishes building a sample.

Precondition

!is_nested() (p. 788). That is, this object must be a sample Builder, not a member Builder.

Postcondition

!is_valid() (p. 788). That is, this Builder can no longer be used.

Returns

The sample, ready to be written.

This function completes and returns the sample built with this Builder.

See also

rti::flat::build_data() (p. 768), the function to create a **FooDataWriter** (p. 1944)-managed sample Builder.

9.331.4.3 add_my_primitive()

```
bool MyFlatMutableBuilder::add_my_primitive (
            int32_t value )
```

Adds a primitive member.

9.331.4.4 add_my_optional_primitive()

```
bool MyFlatMutableBuilder::add_my_optional_primitive (
            int32_t value )
```

Adds a primitive member.

9.331.4.5 add_my_primitive_array()

```
rti::flat::PrimitiveArrayOffset< int32_t, 10 > MyFlatMutableBuilder::add_my_primitive_array ( )
```

Adds a primitive array member.

Returns

The Offset to the array, which can be used to set its values.

9.331.4.6 build_my_primitive_seq()

```
rti::flat::PrimitiveSequenceBuilder< int32_t > MyFlatMutableBuilder::build_my_primitive_seq ( )
```

Begins building a primitive-sequence member.

9.331.4.7 add_my_final()

```
MyFlatFinalOffset MyFlatMutableBuilder::add_my_final ( )
```

Adds a final complex member.

Returns

The Offset to the member, which can be used to set its values.

9.331.4.8 add_my_final_array()

```
rti::flat::FinalAlignedArrayOffset< MyFlatFinalOffset, 10 > MyFlatMutableBuilder::add_my_↩
final_array ( )
```

Adds an array member of final complex elements.

Returns

The Offset to the array, which can be used to set its values.

9.331.4.9 build_my_final_seq()

```
rti::flat::FinalSequenceBuilder< MyFlatFinalOffset > MyFlatMutableBuilder::build_my_final_seq (
)
```

Begins building a sequence member of final complex elements.

9.331.4.10 build_my_mutable()

```
FlatMutableBarBuilder MyFlatMutableBuilder::build_my_mutable ( )
```

Begins building a mutable complex member.

FlatMutableBar represents another arbitrary mutable FlatData type.

9.331.4.11 build_my_mutable_array()

```
rti::flat::MutableArrayBuilder< FlatMutableBarBuilder, 10 > MyFlatMutableBuilder::build_my_↩
mutable_array ( )
```

Begins building an array member of mutable complex elements.

9.331.4.12 build_my_mutable_seq()

```
rti::flat::MutableSequenceBuilder< FlatMutableBarBuilder > MyFlatMutableBuilder::build_my_↩
mutable_seq ( )
```

Begins building a sequence member of mutable complex elements.

9.331.4.13 build_my_string()

```
rti::flat::StringBuilder MyFlatMutableBuilder::build_my_string ( )
```

Begins building a string member.

9.331.4.14 build_my_string_seq()

rti::flat::MutableSequenceBuilder< **rti::flat::StringBuilder** > MyFlatMutableBuilder::build_my_string_seq ( )

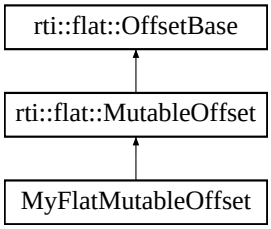
Begins building a sequence member of string elements.

9.332 MyFlatMutableOffset Class Reference

Represents the Offset to an arbitrary user-defined FlatData mutable IDL struct.

#include <Offset.hpp>

Inheritance diagram for MyFlatMutableOffset:



Public Types

- typedef MyFlatMutableConstOffset **ConstOffset**
  The equivalent read-only Offset type.

Public Member Functions

- **MyFlatMutableOffset** ()
  Creates a null Offset.
- int32_t **my_primitive** () const
  Retrieves the value for a primitive member.
- **rti::flat::PrimitiveConstOffset**< int32_t > **my_optional_primitive** () const
  Retrieves a const Offset to an optional primitive.
- const **rti::flat::PrimitiveArrayOffset**< int32_t, 10 > **my_primitive_array** () const
  Retrieves a const Offset to a primitive array.
- const **rti::flat::PrimitiveSequenceOffset**< int32_t > **my_primitive_seq** () const
  Retrieves a const Offset to a primitive sequence.
- **MyFlatFinal::ConstOffset** **my_final** () const
  Retrieves a const Offset to a complex member.
- **rti::flat::FinalAlignedArrayOffset**< **MyFlatFinal::ConstOffset**, 10 > **my_final_array** () const
  Retrieves a const Offset to a complex array.
- **rti::flat::SequenceOffset**< **MyFlatFinal::ConstOffset** > **my_final_seq** () const

Retrieves a const Offset to a complex sequence.

- FlatMutableBar::ConstOffset **my_mutable** () const

  Retrieves a const Offset to a complex member.
- **rti::flat::MutableArrayOffset**< FlatMutableBar::ConstOffset, 10 > **my_mutable_array** () const

  Retrieves a const Offset to a complex array.
- **rti::flat::SequenceOffset**< FlatMutableBar::ConstOffset > **my_mutable_seq** () const

  Retrieves a const Offset to a complex sequence.
- const **rti::flat::StringOffset** **my_string** () const

  Retrieves a const Offset to a string.
- **rti::flat::SequenceOffset**< const **rti::flat::StringOffset** > **my_string_seq** () const

  Retrieves a const Offset to a sequence of strings.
- bool **my_primitive** (int32_t value)

  Sets the value of a primitive member.
- **rti::flat::PrimitiveArrayOffset**< int32_t, 10 > **my_primitive_array** ()

  Retrieves a non-const Offset to a primitive array.
- **rti::flat::PrimitiveSequenceOffset**< int32_t > **my_primitive_seq** ()

  Retrieves a non-const Offset to a primitive sequence.
- **MyFlatFinal::Offset** **my_final** ()

  Retrieves a non-const Offset to a complex member.
- **rti::flat::FinalAlignedArrayOffset**< **MyFlatFinal::Offset**, 10 > **my_final_array** ()

  Retrieves a non-const Offset to a complex array.
- **rti::flat::SequenceOffset**< **MyFlatFinal::Offset** > **my_final_seq** ()

  Retrieves a non-const Offset to a complex sequence.
- FlatMutableBar::Offset **my_mutable** ()

  Retrieves a non-const Offset to a complex member.
- **rti::flat::MutableArrayOffset**< FlatMutableBar::Offset, 10 > **my_mutable_array** ()

  Retrieves a non-const Offset to a complex array.
- **rti::flat::SequenceOffset**< FlatMutableBar::Offset > **my_mutable_seq** ()

  Retrieves a non-const Offset to a complex sequence.
- **rti::flat::StringOffset** **my_string** ()

  Retrieves a non-const Offset to a string.
- **rti::flat::SequenceOffset**< **rti::flat::StringOffset** > **my_string_seq** ()

  Retrieves a non-const Offset to a sequence of strings.

9.332.1 Detailed Description

Represents the Offset to an arbitrary user-defined FlatData mutable IDL struct.

This example type represents the Offset type that **rtiddsgen** would generate for **MyFlatMutable** (p. 770).

It provides accessors for each of its members. Accessors can return other Offsets or primitive values.

9.332.2 Member Typedef Documentation

9.332.2.1 ConstOffset

```
typedef MyFlatMutableConstOffset  MyFlatMutableOffset::ConstOffset
```

The equivalent read-only Offset type.

Each Offset for a user type has an equivalent const Offset that doesn't allow modifying the underlying Sample.

9.332.3 Constructor & Destructor Documentation

9.332.3.1 MyFlatMutableOffset()

```
MyFlatMutableOffset::MyFlatMutableOffset ( )  [inline]
```

Creates a null Offset.

Postcondition

is_null() (p. 2128)

9.332.4 Member Function Documentation

9.332.4.1 my_primitive() [1/2]

```
int32_t MyFlatMutableOffset::my_primitive ( ) const
```

Retrieves the value for a primitive member.

Returns

The value of the member or its default value if this member doesn't exist in this Sample.

9.332.4.2 my_optional_primitive()

```
rti::flat::PrimitiveConstOffset< int32_t > MyFlatMutableOffset::my_optional_primitive ( ) const
```

Retrieves a const Offset to an optional primitive.

Unlike the non-optional **my_primitive()** (p. 2027), which accesses the integer directly, for an optional primitive it is possible to check whether it exists or not. If it doesn't exist, the Offset this function returns will be null (**is_null()** (p. 2128)).

9.332.4.3 my_primitive_array() [1/2]

```
const rti::flat::PrimitiveArrayOffset< int32_t, 10 > MyFlatMutableOffset::my_primitive_array ( )
const
```

Retrieves a const Offset to a primitive array.

9.332.4.4 my_primitive_seq() [1/2]

```
const rti::flat::PrimitiveSequenceOffset< int32_t > MyFlatMutableOffset::my_primitive_seq ( )
const
```

Retrieves a const Offset to a primitive sequence.

9.332.4.5 my_final() [1/2]

```
MyFlatFinal::ConstOffset MyFlatMutableOffset::my_final ( ) const
```

Retrieves a const Offset to a complex member.

9.332.4.6 my_final_array() [1/2]

```
rti::flat::FinalAlignedArrayOffset< MyFlatFinal::ConstOffset, 10 > MyFlatMutableOffset::my_↩
final_array ( ) const
```

Retrieves a const Offset to a complex array.

9.332.4.7 my_final_seq() [1/2]

```
rti::flat::SequenceOffset< MyFlatFinal::ConstOffset > MyFlatMutableOffset::my_final_seq ( )
const
```

Retrieves a const Offset to a complex sequence.

9.332.4.8 my_mutable() [1/2]

```
FlatMutableBar::ConstOffset MyFlatMutableOffset::my_mutable ( ) const
```

Retrieves a const Offset to a complex member.

FlatMutableBar is another arbitrary user-defined mutable FlatData type.

9.332.4.9 my_mutable_array() [1/2]

```
rti::flat::MutableArrayOffset< FlatMutableBar::ConstOffset, 10 > MyFlatMutableOffset::my_↩
mutable_array ( ) const
```

Retrieves a const Offset to a complex array.

9.332.4.10 my_mutable_seq() [1/2]

```
rti::flat::SequenceOffset< FlatMutableBar::ConstOffset > MyFlatMutableOffset::my_mutable_seq ( )
const
```

Retrieves a const Offset to a complex sequence.

9.332.4.11 my_string() [1/2]

```
const rti::flat::StringOffset MyFlatMutableOffset::my_string ( ) const
```

Retrieves a const Offset to a string.

9.332.4.12 my_string_seq() [1/2]

```
rti::flat::SequenceOffset< const rti::flat::StringOffset > MyFlatMutableOffset::my_string_seq (
) const
```

Retrieves a const Offset to a sequence of strings.

9.332.4.13 my_primitive() [2/2]

```
bool MyFlatMutableOffset::my_primitive (
            int32_t value )
```

Sets the value of a primitive member.

9.332.4.14 my_primitive_array() [2/2]

```
rti::flat::PrimitiveArrayOffset< int32_t, 10 > MyFlatMutableOffset::my_primitive_array ( )
```

Retrieves a non-const Offset to a primitive array.

9.332.4.15 my_primitive_seq() [2/2]

```
rti::flat::PrimitiveSequenceOffset< int32_t > MyFlatMutableOffset::my_primitive_seq ( )
```

Retrieves a non-const Offset to a primitive sequence.

9.332.4.16 my_final() [2/2]

```
MyFlatFinal::Offset MyFlatMutableOffset::my_final ( )
```

Retrieves a non-const Offset to a complex member.

9.332.4.17 my_final_array() [2/2]

```
rti::flat::FinalAlignedArrayOffset< MyFlatFinal::Offset, 10 > MyFlatMutableOffset::my_final_↩
array ( )
```

Retrieves a non-const Offset to a complex array.

9.332.4.18 my_final_seq() [2/2]

```
rti::flat::SequenceOffset< MyFlatFinal::Offset > MyFlatMutableOffset::my_final_seq ( )
```

Retrieves a non-const Offset to a complex sequence.

9.332.4.19 my_mutable() [2/2]

```
FlatMutableBar::Offset MyFlatMutableOffset::my_mutable ( )
```

Retrieves a non-const Offset to a complex member.

FlatMutableBar is another arbitrary user-defined mutable FlatData type.

9.332.4.20 my_mutable_array() [2/2]

```
rti::flat::MutableArrayOffset< FlatMutableBar::Offset, 10 > MyFlatMutableOffset::my_mutable_↩
array ( )
```

Retrieves a non-const Offset to a complex array.

9.332.4.21 my_mutable_seq() [2/2]

```
rti::flat::SequenceOffset< FlatMutableBar::Offset > MyFlatMutableOffset::my_mutable_seq ( )
```

Retrieves a non-const Offset to a complex sequence.

9.332.4.22 my_string() [2/2]

```
rti::flat::StringOffset MyFlatMutableOffset::my_string ( )
```

Retrieves a non-const Offset to a string.

9.332.4.23 my_string_seq() [2/2]

```
rti::flat::SequenceOffset< rti::flat::StringOffset > MyFlatMutableOffset::my_string_seq ( )
```

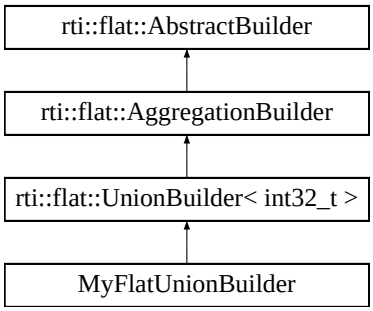
Retrieves a non-const Offset to a sequence of strings.

9.333 MyFlatUnionBuilder Class Reference

Represents the Builder for an arbitrary user-defined mutable IDL union.

`#include <Builder.hpp>`

Inheritance diagram for MyFlatUnionBuilder:



Public Types

- typedef **MyFlatUnionOffset Offset**
  The related offset type.

Public Member Functions

- **MyFlatUnionBuilder** ()
  Creates an invalid Builder.
- **Offset finish** ()
  Finishes building a member.
- **MyFlatUnion** * **finish_sample** ()
  Finishes building a sample.
- bool **add_my_primitive** (int32_t value)
  Adds a primitive member.
- **MyFlatMutableBuilder build_my_mutable** (int32_t discriminator=1)
  Builds a mutable-struct member.
- **MyFlatFinal::Offset add_my_final** ()
  Adds a final-struct member.

Additional Inherited Members

9.333.1 Detailed Description

Represents the Builder for an arbitrary user-defined mutable IDL union.

This example type represents the **Builder** (p. 766) type that **rtiddsgen** would generate for **MyFlatUnion** (p. 770) and allows creating a sample or a member of that type.

Union builders are similar to struct builders (see **MyFlatMutableBuilder** (p. 2018)), except that they only allow adding/building **one member**.

"Add" and "build" functions automatically set the discriminator value that in the IDL definition selects that member. If more than one discriminator value selects a member, the add/build function allows picking one.

9.333.2 Member Typedef Documentation

9.333.2.1 Offset

```
typedef MyFlatUnionOffset MyFlatUnionBuilder::Offset
```

The related offset type.

9.333.3 Constructor & Destructor Documentation

9.333.3.1 MyFlatUnionBuilder()

```
MyFlatUnionBuilder::MyFlatUnionBuilder ( ) [inline]
```

Creates an invalid Builder.

Postcondition

!is_valid()

Note

Top-level builders are created with **rti::flat::build_data()** (p. 768), and member builders are created with the corresponding `build_<member>` function.

9.333.4 Member Function Documentation

9.333.4.1 finish()

```
Offset MyFlatUnionBuilder::finish ( )
```

Finishes building a member.

See also

MyMutableBuilder::finish()

9.333.4.2 finish_sample()

```
MyFlatUnion * MyFlatUnionBuilder::finish_sample ( )
```

Finishes building a sample.

See also

MyMutableBuilder::finish_sample()

9.333.4.3 add_my_primitive()

```
bool MyFlatUnionBuilder::add_my_primitive (
            int32_t value )
```

Adds a primitive member.

This function automatically selects the discriminator 0, which corresponds to the member 'my_primitive' (see **MyFlat↩ Union** (p. 770))

9.333.4.4 build_my_mutable()

```
MyFlatMutableBuilder MyFlatUnionBuilder::build_my_mutable (
            int32_t discriminator = 1 )
```

Builds a mutable-struct member.

Parameters

discriminator	Allows selecting one of the possible discriminator values for this member: 1 or 2. This argument is optional: if not specified it selects the first 'case' label in the IDL definition (1).

Returns

The Builder to build this member

9.333.4.5 add_my_final()

```
MyFlatFinal::Offset MyFlatUnionBuilder::add_my_final ( )
```

Adds a final-struct member.

Returns

The Offset to the member, which can be used to set its values.

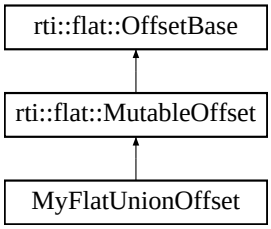
This function automatically selects the discriminator value 3, which corresponds to the member 'my_final' (see **MyFlat↩ Union** (p. 770)).

9.334 MyFlatUnionOffset Class Reference

Represents the Offset to an arbitrary user-defined FlatData mutable IDL union.

`#include <Offset.hpp>`

Inheritance diagram for MyFlatUnionOffset:



Public Types

- typedef MyFlatUnionConstOffset **ConstOffset**

  The equivalent read-only Offset type.

Public Member Functions

- **MyFlatUnionOffset** ()

  Creates a null Offset.
- int32_t **_d** () const

  Retrieves the union discriminator.
- int32_t **my_primitive** () const

  Retrieves the value for a primitive member.
- **MyFlatMutable::ConstOffset my_mutable** () const

  Retrieves a const Offset to a complex member.
- **MyFlatFinal::ConstOffset my_final** () const

  Retrieves a const Offset to a complex member.
- bool **my_primitive** (int32_t value)

  Sets the value for a primitive member.
- **MyFlatMutable::Offset my_mutable** ()

  Retrieves a non-const Offset to a complex member.
- **MyFlatFinal::Offset my_final** ()

  Retrieves a non-const Offset to a complex member.

9.334.1 Detailed Description

Represents the Offset to an arbitrary user-defined FlatData mutable IDL union.

This example type represents the Offset type that **rtiddsgen** would generate for **MyFlatUnion** (p. 770).

It provides accessors for each of its members, plus the discriminator **_d()** (p. 2036). Accessors can return other Offsets or primitive values.

Given a union, only one member, the one identified by **_d()** (p. 2036), can exist at a time. The discriminator cannot be modified.

9.334.2 Member Typedef Documentation

9.334.2.1 ConstOffset

```
typedef MyFlatUnionConstOffset  MyFlatUnionOffset::ConstOffset
```

The equivalent read-only Offset type.

9.334.3 Constructor & Destructor Documentation

9.334.3.1 MyFlatUnionOffset()

```
MyFlatUnionOffset::MyFlatUnionOffset ( )  [inline]
```

Creates a null Offset.

Postcondition

is_null() (p. 2128)

9.334.4 Member Function Documentation

9.334.4.1 _d()

```
int32_t MyFlatUnionOffset::_d ( ) const
```

Retrieves the union discriminator.

Returns

The union discriminator, which identifies which field this union contains.

In this example:

- 0 selects **my_primitive()** (p. 2037)
- 1 and 2 select **my_mutable()** (p. 2038)
- 3 selects my_final

Any other discriminator value indicates that no member or an unknown member follows.

Note that the discriminator cannot be modified, since that would potentially change the size of this Sample by selecting a different member. The discriminator is selected during the building of a Sample (see MyUnionBuilder).

9.334.4.2 my_primitive() [1/2]

```
int32_t MyFlatUnionOffset::my_primitive ( ) const
```

Retrieves the value for a primitive member.

Returns

The value of my_primitive if this member is selected by **_d()** (p. 2036)
or its default value otherwise.

9.334.4.3 my_mutable() [1/2]

```
MyFlatMutable::ConstOffset MyFlatUnionOffset::my_mutable ( ) const
```

Retrieves a const Offset to a complex member.

Returns

The Offset to the 'my_mutable' member if selected by **_d()** (p. 2036), or a **null Offset** (p. 773) otherwise

9.334.4.4 my_final() [1/2]

```
MyFlatFinal::ConstOffset MyFlatUnionOffset::my_final ( ) const
```

Retrieves a const Offset to a complex member.

Returns

The Offset to the 'my_final' member if selected by **_d()** (p. 2036), or a **null Offset** (p. 773) otherwise

9.334.4.5 my_primitive() [2/2]

```
bool MyFlatUnionOffset::my_primitive (
            int32_t value )
```

Sets the value for a primitive member.

Precondition

_d() (p. 2036) must select 'my_primitive', otherwise this function returns false.

Returns

true if my_primitive was set or false if the operation failed.

9.334.4.6 my_mutable() [2/2]

```
MyFlatMutable::Offset MyFlatUnionOffset::my_mutable ( )
```

Retrieves a non-const Offset to a complex member.

Returns

The Offset to the 'my_mutable' member if selected by **_d()** (p. 2036), or a **null Offset** (p. 773) otherwise

9.334.4.7 my_final() [2/2]

```
MyFlatFinal::Offset MyFlatUnionOffset::my_final ( )
```

Retrieves a non-const Offset to a complex member.

Returns

The Offset to the 'my_final' member if selected by **_d()** (p. 2036), or a **null Offset** (p. 773) otherwise

9.335 NDDS_Config_LibraryVersion_t Struct Reference

The version of a single library shipped as part of an RTI Connext distribution.

Public Attributes

- **DDS_Long major**

  The major version of a single RTI Connext library.
- **DDS_Long minor**

  The minor version of a single RTI Connext library.
- char **release**

  The release letter of a single RTI Connext library.
- **DDS_Long build**

  The build number of a single RTI Connext library.

9.335.1 Detailed Description

The version of a single library shipped as part of an RTI Connext distribution.

RTI Connext is comprised of a number of separate libraries. Although RTI Connext as a whole has a version, the individual libraries each have their own versions as well. It may be necessary to check these individual library versions when seeking technical support.

9.335.2 Member Data Documentation

9.335.2.1 major

```
DDS_Long NDDS_Config_LibraryVersion_t::major
```

The major version of a single RTI Connext library.

9.335.2.2 minor

```
DDS_Long NDDS_Config_LibraryVersion_t::minor
```

The minor version of a single RTI Connext library.

9.335.2.3 release

```
char NDDS_Config_LibraryVersion_t::release
```

The release letter of a single RTI Connext library.

9.335.2.4 build

```
DDS_Long NDDS_Config_LibraryVersion_t::build
```

The build number of a single RTI Connext library.

9.336 NDDS_Config_LogMessage Struct Reference

A log message's input parameter to a logging device, including the text and additional information.

Public Attributes

- const char * **text**

  Message text.
- **NDDS_Config_LogLevel level**

  Message level.
- **DDS_Boolean is_security_message**

  Indicates if the message is a security-related message.
- **DDS_Boolean is_license_message**

  Indicates if the message is a license-related message.
- **DDS_UnsignedLong message_id**

  A numeric code that identifies a specific log message.
- struct **DDS_Duration_t timestamp**

  The time when the log message was printed.
- **NDDS_Config_LogFacility facility**

  *The Facility associated with the log message. See **NDDS_Config_LogFacility** (p. 727).*

9.336.1 Detailed Description

A log message's input parameter to a logging device, including the text and additional information.

9.336.2 Member Data Documentation

9.336.2.1 text

```
const char* NDDS_Config_LogMessage::text
```

Message text.

9.336.2.2 level

```
NDDS_Config_LogLevel NDDS_Config_LogMessage::level
```

Message level.

9.336.2.3 is_security_message

```
DDS_Boolean NDDS_Config_LogMessage::is_security_message
```

Indicates if the message is a security-related message.

A "security-related message" is a log message that meets any of the following:

- A security event logged with the RTI Security Plugins Logging Plugin.
- Any regular message coming from the RTI Security Plugins.
- RTI TLS Support log messages related to OpenSSL (e.g., SSL handshake failures or certificate validation failures).

9.336.2.4 is_license_message

```
DDS_Boolean NDDS_Config_LogMessage::is_license_message
```

Indicates if the message is a license-related message.

When using a licensed version of RTI Connext (such as the RTI Connext Express edition), certain messages related to the license, such as the trial expiration date or license limitations, are logged when a DomainParticipant is created. This flag indicates whether the message is one of those license-related messages.

9.336.2.5 message_id

DDS_UnsignedLong NDDS_Config_LogMessage::message_id

A numeric code that identifies a specific log message.

Two log messages that have the same message_id will have a similar structure. E.g. "ERROR: Failed to get Data↩ WriterQos" and "ERROR: Failed to get TopicName". In this case, the message_id is associated to the get failure.

9.336.2.6 timestamp

struct **DDS_Duration_t** NDDS_Config_LogMessage::timestamp

The time when the log message was printed.

9.336.2.7 facility

NDDS_Config_LogFacility NDDS_Config_LogMessage::facility

The Facility associated with the log message. See **NDDS_Config_LogFacility** (p. 727).

9.337 NDDS_Transport_Address_t Struct Reference

Addresses are stored individually as network-ordered bytes.

#include <transport_common_user.h>

Public Attributes

- unsigned char **network_ordered_value** [NDDS_TRANSPORT_ADDRESS_LENGTH]

9.337.1 Detailed Description

Addresses are stored individually as network-ordered bytes.

RTI Connext addresses are numerically stored in a transport independent manner. RTI Connext uses a IPv6-compatible format, which means that the data structure to hold an **NDDS_Transport_Address_t** (p. 2042) is the same size as a data structure needed to hold an IPv6 address.

In addition, the functions provided to translate a string representation of an RTI Connext address to a value assumes that the string presentation follows the IPv6 address presentation as specified in RFC 2373.

An **NDDS_Transport_Address_t** (p. 2042) always stores the address in network-byte order (which is Big Endian).

For example, IPv4 multicast address of 225.0.0.0 is represented by

{{0,0,0,0, 0,0,0,0, 0,0,0,0, 0xE1,0,0,0}} regardless of endianness,

where 0xE1 is the 13th byte of the structure (network_ordered_value[12]).

9.337.2 Member Data Documentation

9.337.2.1 network_ordered_value

```
unsigned char NDDS_Transport_Address_t::network_ordered_value[NDDS_TRANSPORT_ADDRESS_LENGTH]
```

network-byte ordered (i.e., bit 0 is the most significant bit and bit 128 is the least significant bit).

9.338 NDDS_Transport_Interface_t Struct Reference

Storage for the description of a network interface used by a Transport Plugin.

```
#include <transport_common.h>
```

Public Attributes

- **NDDS_Transport_ClassId_t transport_classid**

  The transport classid of the interface.
- **NDDS_Transport_Address_t address**

  An unicast address that uniquely identifies this interface in the network specified by the transport class.
- **NDDS_Transport_Interface_Status_t status**

  The state of the interface.
- RTI_UINT16 **rank**

  Rank of the interface. Used when allow_interfaces_list Qos is set. A rank value will be assigned to each of the interfaces that match the allow_interfaces_list filter allowing an endpoint to prioritize some interfaces upon others.

9.338.1 Detailed Description

Storage for the description of a network interface used by a Transport Plugin.

9.338.2 Member Data Documentation

9.338.2.1 transport_classid

```
NDDS_Transport_ClassId_t NDDS_Transport_Interface_t::transport_classid
```

The transport classid of the interface.

9.338.2.2 address

NDDS_Transport_Address_t NDDS_Transport_Interface_t::address

An unicast address that uniquely identifies this interface in the network specified by the transport class.

9.338.2.3 status

NDDS_Transport_Interface_Status_t NDDS_Transport_Interface_t::status

The state of the interface.

9.338.2.4 rank

RTI_UINT16 NDDS_Transport_Interface_t::rank

Rank of the interface. Used when allow_interfaces_list Qos is set. A rank value will be assigned to each of the interfaces that match the allow_interfaces_list filter allowing an endpoint to prioritize some interfaces upon others.

9.339 NDDS_Transport_Property_t Struct Reference

Base configuration structure that must be inherited by derived Transport Plugin classes.

`#include <transport_interface_user.h>`

Public Attributes

- **NDDS_Transport_ClassId_t classid**

  The Transport-Plugin Class ID.
- RTI_INT32 **address_bit_count**

  Number of bits in a 16-byte address that are used by the transport. Should be between -128 and 128.
- RTI_INT32 **properties_bitmap**

  A bitmap that defines various properties of the transport to the RTI Connext core.
- RTI_INT32 **gather_send_buffer_count_max**

  Specifies the maximum number of buffers that RTI Connext can pass to the `send()` method of a transport plugin.
- RTI_INT32 **message_size_max**

  The maximum size of an RTPS message in bytes that can be sent or received by the transport plugin.
- char ** **allow_interfaces_list**

  A list of strings, each identifying a range of interface addresses or an interface name. If the list is non-empty (i.e., `allow_interfaces_list_length` > 0), allow the use of only these interfaces. If the list is empty, allow the use of all interfaces.

- RTI_INT32 **allow_interfaces_list_length**

  Number of elements in the `allow_interfaces_list`*.*
- char ** **deny_interfaces_list**

  A list of strings, each identifying a range of interface addresses or an interface name. If the list is non-empty (i.e., `deny_interfaces_list_length` *> 0), deny the use of these interfaces.*
- RTI_INT32 **deny_interfaces_list_length**

  Number of elements in the `deny_interfaces_list`*.*
- char ** **allow_multicast_interfaces_list**

  A list of strings, each identifying a range of interface addresses or an interface name. If the list is non-empty (i.e., `allow_multicast_interfaces_list_length` *> 0), allow the use of multicast only on these interfaces; otherwise allow the use of all the allowed interfaces.*
- RTI_INT32 **allow_multicast_interfaces_list_length**

  Number of elements in the `allow_multicast_interfaces_list`*.*
- char ** **deny_multicast_interfaces_list**

  A list of strings, each identifying a range of interface addresses or an interface name. If the list is non-empty (i.e., `deny_multicast_interfaces_list_length` *> 0), deny the use of those interfaces for multicast.*
- RTI_INT32 **deny_multicast_interfaces_list_length**

  Number of elements in `deny_multicast_interfaces_list`*.*
- struct **NDDS_Transport_UUID transport_uuid**

  Univocally identifies a transport plugin.
- char * **thread_name_prefix**

  Prefix used to name the transport threads.
- RTI_INT32 **max_interface_count**

  Maximum number of addresses from the `allow_interfaces_list` *that will be announced.*

9.339.1 Detailed Description

Base configuration structure that must be inherited by derived Transport Plugin classes.

This structure contains properties that must be set before registration of any transport plugin with RTI Connext. The RTI Connext core will configure itself to use the plugin based on the properties set within this structure.

A transport plugin may extend from this structure to add transport-specific properties.

In the C-language, this can be done by creating a custom plugin property structure whose first member is a **NDDS_Transport_Property_t** (p. 2044) structure.

For example,

```
struct MyTransport_Plugin_Property_t {
    NDDS_Transport_Property_t base_properties;
    int myIntProperty;
    < etc >;
};
```

WARNING: The transport properties of an instance of a Transport Plugin should be considered immutable after the plugin has been created. That means the values contained in the property structure stored as a part of the transport plugin itself should not be changed. If those values are modified, the results are undefined.

9.339.2 Member Data Documentation

9.339.2.1 classid

NDDS_Transport_ClassId_t NDDS_Transport_Property_t::classid

The Transport-Plugin Class ID.

The transport plugin sets the value for this field.

Class IDs below **NDDS_TRANSPORT_CLASSID_RESERVED_RANGE** (p. 446) are reserved for RTI (Real-Time Innovations) usage.

User-defined transports should set an ID above this range.

The ID should be globally unique for each Transport-Plugin class. Transport-Plugin implementors should ensure that the class IDs do not conflict with each other amongst different Transport-Plugin classes.

Invariant

The `classid` is invariant for the lifecycle of a transport plugin.

9.339.2.2 address_bit_count

RTI_INT32 NDDS_Transport_Property_t::address_bit_count

Number of bits in a 16-byte address that are used by the transport. Should be between -128 and 128.

The transport plugin sets the value for this field.

A transport plugin should define the range of addresses that are meaningful to the plugin. It does this by setting the number of bits of an IPv6 address that will be used to designate an address in the network to which the transport plugin is connected. The sign of this field determines whether the count of bits starts on the most significant bit (negative sign) or on the least significant bit (positive sign).

For example, for an address range of 0-255, the address_bit_count should be set to 8. For the range of addresses used by IPv4 (4 bytes), it should be set to 32.

See also

Transport Class Attributes (p. 174)

9.339.2.3 properties_bitmap

```
RTI_INT32 NDDS_Transport_Property_t::properties_bitmap
```

A bitmap that defines various properties of the transport to the RTI Connext core.

Currently, the only property supported is whether or not the transport plugin will always loan a buffer when RTI Connext tries to receive a message using the plugin. This is in support of a zero-copy interface.

See also

NDDS_TRANSPORT_PROPERTY_BIT_BUFFER_ALWAYS_LOANED (p. 446)

9.339.2.4 gather_send_buffer_count_max

```
RTI_INT32 NDDS_Transport_Property_t::gather_send_buffer_count_max
```

Specifies the maximum number of buffers that RTI Connext can pass to the `send()` method of a transport plugin.

The transport plugin `send()` API supports a gather-send concept, where the `send()` call can take several discontiguous buffers, assemble and send them in a single message. This enables RTI Connext to send a message from parts obtained from different sources without first having to copy the parts into a single contiguous buffer.

However, most transports that support a gather-send concept have an upper limit on the number of buffers that can be gathered and sent. Setting this value will prevent RTI Connext from trying to gather too many buffers into a send call for the transport plugin.

RTI Connext requires all transport-plugin implementations to support a gather-send of least a minimum number of buffers. This minimum number is defined to be **NDDS_TRANSPORT_PROPERTY_GATHER_SEND_BUFFER_↩ COUNT_MIN** (p. 447).

If the underlying transport does not support a gather-send concept directly, then the transport plugin itself must copy the separate buffers passed into the `send()` call into a single buffer for sending or otherwise send each buffer individually. However this is done by the transport plugin, the receive_rEA() call of the destination application should assemble, if needed, all of the pieces of the message into a single buffer before the message is passed to the RTI Connext layer.

9.339.2.5 message_size_max

```
RTI_INT32 NDDS_Transport_Property_t::message_size_max
```

The maximum size of an RTPS message in bytes that can be sent or received by the transport plugin.

If the maximum size of a message that can be sent by a transport plugin is user configurable, the transport plugin should provide a default value for this property. In any case, this value must be set before the transport plugin is registered, so that RTI Connext can properly use the plugin.

For information about the default value for different transports, see the `Core Libraries User's Manual`.

9.339.2.6 allow_interfaces_list

```
char** NDDS_Transport_Property_t::allow_interfaces_list
```

A list of strings, each identifying a range of interface addresses or an interface name. If the list is non-empty (i.e., `allow_interfaces_list_length` > 0), allow the use of only these interfaces. If the list is empty, allow the use of all interfaces.

The "white" list restricts *reception* to a particular set of interfaces for unicast UDP.

For UDPv4 and UDPv6 transports, the resulting list also controls the interfaces over which the DomainParticipant will send multicast traffic to the remote DomainParticipants (if multicast is supported on the platform).

Note: This property does not affect the interfaces that the transport uses to send unicast data from that Domain↩ Participant. That decision is made by the OS based on the destination address.

It is up to the transport plugin to interpret the list of strings passed in.

For example, the following are acceptable strings in IPv4 format: 192.168.1.1, 192.168.1.∗, 192.168.∗, 192.∗, ether0

The left-to-right order of this list matters if you are using the max_interface_count to limit the allowable interfaces further. See `max_interface_count`.

This property is not interpreted by the RTI Connext core; it is provided merely as a convenient and standardized way to specify the interfaces for the benefit of the transport plugin developer and user.

The caller (user) must manage the memory of the list. The memory may be freed after the **DDSDomainParticipant** (p. 1619) is deleted.

[default] empty list that represents all available interfaces.

See also

NDDS_Transport_Property_t::max_interface_count (p. 2051)

9.339.2.7 allow_interfaces_list_length

```
RTI_INT32 NDDS_Transport_Property_t::allow_interfaces_list_length
```

Number of elements in the `allow_interfaces_list`.

By default, `allow_interfaces_list_length` = 0, i.e. an empty list.

This property is not interpreted by the RTI Connext core; it is provided merely as a convenient and standardized way to specify the interfaces for the benefit of the transport plugin developer and user.

[default] 0

9.339.2.8 deny_interfaces_list

```
char** NDDS_Transport_Property_t::deny_interfaces_list
```

A list of strings, each identifying a range of interface addresses or an interface name. If the list is non-empty (i.e., `deny_interfaces_list_length` > 0), deny the use of these interfaces.

This "black" list is applied *after* the `allow_interfaces_list` and filters out the interfaces that should not be used.

The resulting list restricts *reception* to a particular set of interfaces for unicast UDP.

For UDPv4 and UDPv6 transports, the resulting list also controls the interfaces over which the DomainParticipant will send multicast traffic to the remote DomainParticipants (if multicast is supported on the platform).

Note: This property does not affect the interfaces that the transport uses to send unicast data from that Domain↩ Participant. That decision is made by the OS based on the destination address.

It is up to the transport plugin to interpret the list of strings passed in.

For example, the following are acceptable strings in IPv4 format: 192.168.1.1, 192.168.1.∗, 192.168.∗, 192.∗, ether0

This property is not interpreted by the RTI Connext core; it is provided merely as a convenient and standardized way to specify the interfaces for the benefit of the transport plugin developer and user.

The caller (user) must manage the memory of the list. The memory may be freed after the **DDSDomainParticipant** (p. 1619) is deleted.

[default] empty list that represents no denied interfaces.

9.339.2.9 deny_interfaces_list_length

```
RTI_INT32 NDDS_Transport_Property_t::deny_interfaces_list_length
```

Number of elements in the `deny_interfaces_list`.

By default, `deny_interfaces_list_length` = 0 (i.e., an empty list).

This property is not interpreted by the RTI Connext core; it is provided merely as a convenient and standardized way to specify the interfaces for the benefit of the transport plugin developer and user.

[default] 0

9.339.2.10 allow_multicast_interfaces_list

```
char** NDDS_Transport_Property_t::allow_multicast_interfaces_list
```

A list of strings, each identifying a range of interface addresses or an interface name. If the list is non-empty (i.↩ e., `allow_multicast_interfaces_list_length` > 0), allow the use of multicast only on these interfaces; otherwise allow the use of all the allowed interfaces.

This "white" list sub-selects from the allowed interfaces obtained *after* applying the `allow_interfaces_list` "white" list *and* the `deny_interfaces_list` "black" list.

After `allow_multicast_interfaces_list`, the `deny_multicast_interfaces_list` is applied. Multicast output will be sent and may be received over the interfaces in the resulting list.

If this list is empty, all the allowed interfaces will be potentially used for multicast. It is up to the transport plugin to interpret the list of strings passed in.

This property is not interpreted by the RTI Connext core; it is provided merely as a convenient and standardized way to specify the interfaces for the benefit of the transport plugin developer and user.

The caller (user) must manage the memory of the list. The memory may be freed after the **DDSDomainParticipant** (p. 1619) is deleted.

[default] empty list that represents all available interfaces.

9.339.2.11 allow_multicast_interfaces_list_length

```
RTI_INT32 NDDS_Transport_Property_t::allow_multicast_interfaces_list_length
```

Number of elements in the `allow_multicast_interfaces_list`.

By default, `allow_multicast_interfaces_list_length` = 0 (i.e., an empty list).

This property is not interpreted by the RTI Connext core; it is provided merely as a convenient and standardized way to specify the interfaces for the benefit of the transport plugin developer and user.

[default] 0

9.339.2.12 deny_multicast_interfaces_list

```
char** NDDS_Transport_Property_t::deny_multicast_interfaces_list
```

A list of strings, each identifying a range of interface addresses or an interface name. If the list is non-empty (i.e., `deny_multicast_interfaces_list_length` > 0), deny the use of those interfaces for multicast.

This "black" list is applied after `allow_multicast_interfaces_list` and filters out interfaces that should not be used for multicast.

Multicast output will be sent and may be received over the interfaces in the resulting list.

It is up to the transport plugin to interpret the list of strings passed in.

This property is not interpreted by the RTI Connext core; it is provided merely as a convenient and standardized way to specify the interfaces for the benefit of the transport plugin developer and user.

The caller (user) must manage the memory of the list. The memory may be freed after the **DDSDomainParticipant** (p. 1619) is deleted.

[default] empty list that represents no denied interfaces.

9.339.2.13 deny_multicast_interfaces_list_length

```
RTI_INT32 NDDS_Transport_Property_t::deny_multicast_interfaces_list_length
```

Number of elements in `deny_multicast_interfaces_list`.

By default, `deny_multicast_interfaces_list_length` = 0 (i.e., an empty list).

This property is not interpreted by the RTI Connext core; it is provided merely as a convenient and standardized way to specify the interfaces for the benefit of the transport plugin developer and user.

[default] 0

9.339.2.14 transport_uuid

```
struct NDDS_Transport_UUID NDDS_Transport_Property_t::transport_uuid
```

Univocally identifies a transport plugin.

It represents the prefix of the participant GUID.

9.339.2.15 thread_name_prefix

```
char* NDDS_Transport_Property_t::thread_name_prefix
```

Prefix used to name the transport threads.

This field is optional.

The maximum size of this string is 8 characters.

If "thread_name_prefix" is not set by the user, RTI Connext creates the following prefix: 'r' + 'Tr' + participant identifier + '\0'.

Where 'r' indicates this is a thread from RTI, 'Tr' indicates the thread is related to a transport, and participant identifier contains 5 characters as follows:

If participant_name is set: The participant identifier will be the first 3 characters and the last 2 characters of the participant_name.

If participant_name is not set, then the identifier is computed as domain_id (3 characters) followed by participant_id (2 characters).

If participant_name is not set and the participant_id is set to -1 (defalt value), then the participant identifier is computed as the last 5 digits of the rtps_instance_id in the participant GUID.

9.339.2.16 max_interface_count

```
RTI_INT32 NDDS_Transport_Property_t::max_interface_count
```

Maximum number of addresses from the `allow_interfaces_list` that will be announced.

By default, `max_interface_count` = LENGTH_UNLIMITED: all the interfaces are announced.

This feature is useful if you want to control the network interfaces on which your DomainParticipants receive data. For example, if you have one wired and one wireless interface both up and running, and `max_interface_count` is set to 1, the DomainParticipant will receive data over the interface you list first in the `allow_interfaces_list` -for example, the wired one.

RTI Connext selects the preferred interface(s) by iterating over the list of allowed interfaces until the first `max_↩ interfaces_count` of active interfaces encountered are announced. The order of iteration is left to right as specified in the `allow_interfaces_list` setting.

This setting applies only if the `allow_interfaces_list` is not empty.

The `max_interface_count` setting does not consider end-to-end connectivity to select interfaces. The decision is based purely on whether interfaces are up or down in a node. Therefore, this feature is not intended to be used in the following scenarios:

- A DomainParticipant is not reachable by other DomainParticipants in all the interfaces in the `allow_↩ interfaces_list`. This could occur if the DomainParticipant is in different subnets, and some of these subnets cannot be reached by other DomainParticipants.
- End-to-end connectivity issues lead to situations in which the interfaces selected after applying `max_↩ interface_count` cannot be reached by other DomainParticipants.

For UDPv4 and UDPv6 transports, this feature also affects multicast traffic by limiting the interfaces over which a DomainParticipant sends multicast traffic. The (allow/deny)_multicast_interfaces_list applies to the interfaces selected by using the `max_interfaces_count` property.

Note: If a pattern string in the `allow_interfaces_list` matches multiple interface addresses, and `max_↩ interface_count` is set to a finite value, the order for the matching allowed interfaces is decided based on the order in which the operating system provides these interfaces.

[default] LENGTH_UNLIMITED

See also

NDDS_Transport_Property_t::allow_interfaces_list (p. 2047)

9.340 NDDS_Transport_Shmem_Property_t Struct Reference

Subclass of **NDDS_Transport_Property_t** (p. 2044) allowing specification of parameters that are specific to the shared-memory transport.

```
#include <transport_shmem.h>
```

Public Attributes

- struct **NDDS_Transport_Property_t parent**

  Generic properties of all transport plugins.
- RTI_INT32 **received_message_count_max**

  Number of messages that can be buffered in the receive queue.
- RTI_INT32 **receive_buffer_size**

  The total number of bytes that can be buffered in the receive queue.
- RTIBool **enable_udp_debugging**

  Enables UDP debugging when using shared memory.
- **NDDS_Transport_Address_t udp_debugging_address**

  *IP address to which shared memory traffic will be published if **NDDS_Transport_Shmem_Property_t::enable_udp_debugging** (p. 2054) is set to '1'.*
- **NDDS_Transport_Port_t udp_debugging_port**

  *Port to which shared memory traffic will be published if **NDDS_Transport_Shmem_Property_t::enable_udp_debugging** (p. 2054) is set to '1'.*

9.340.1 Detailed Description

Subclass of **NDDS_Transport_Property_t** (p. 2044) allowing specification of parameters that are specific to the shared-memory transport.

See also

NDDSTransportSupport::set_builtin_transport_property() (p. 2109)

9.340.2 Member Data Documentation

9.340.2.1 parent

`struct` **`NDDS_Transport_Property_t`** `NDDS_Transport_Shmem_Property_t::parent`

Generic properties of all transport plugins.

9.340.2.2 received_message_count_max

`RTI_INT32 NDDS_Transport_Shmem_Property_t::received_message_count_max`

Number of messages that can be buffered in the receive queue.

This does not guarantee that the Transport-Plugin will actually be able to buffer `received_message_count_max` messages of the maximum size set in **NDDS_Transport_Property_t::message_size_max** (p. 2047). The total number of bytes that can be buffered for a transport plug-in is actually controlled by `receive_buffer_size`.

See also

NDDS_Transport_Property_t (p. 2044), **NDDS_TRANSPORT_SHMEM_RECEIVED_MESSAGE_COUNT_MAX_DEFAULT** (p. 461)

9.340.2.3 receive_buffer_size

```
RTI_INT32 NDDS_Transport_Shmem_Property_t::receive_buffer_size
```

The total number of bytes that can be buffered in the receive queue.

This number controls how much memory is allocated by the plugin for the receive queue. The actual number of bytes allocated is:

```
size = receive_buffer_size + message_size_max +
       received_message_count_max * fixedOverhead
```

where `fixedOverhead` is some small number of bytes used by the queue data structure. The following rules are noted:

- `receive_buffer_size` < `message_size_max` * `received_message_count_max`, then the transport plugin will not be able to store `received_message_count_max` messages of size `message_size_max`.

- `receive_buffer_size` > `message_size_max` * `received_message_count_max`, then there will be memory allocated that cannot be used by the plugin and thus wasted.

To optimize memory usage, the user is allowed to specify a size for the receive queue to be less than that required to hold the maximum number of messages which are all of the maximum size.

In most situations, the average message size may be far less than the maximum message size. So for example, if the maximum message size is 64 K bytes, and the user configures the plugin to buffer at least 10 messages, then 640 K bytes of memory would be needed if all messages were 64 K bytes. Should this be desired, then `receive_buffer_size` should be set to 640 K bytes.

However, if the average message size is only 10 K bytes, then the user could set the `receive_buffer_size` to 100 K bytes. This allows the user to optimize the memory usage of the plugin for the average case and yet allow the plugin to handle the extreme case.

NOTE, the queue will always be able to hold 1 message of `message_size_max` bytes, no matter what the value of receive_buffer_size is.

See also

NDDS_TRANSPORT_SHMEM_RECEIVE_BUFFER_SIZE_DEFAULT (p. 461)

9.340.2.4 enable_udp_debugging

```
RTIBool NDDS_Transport_Shmem_Property_t::enable_udp_debugging
```

Enables UDP debugging when using shared memory.

If set to '1', all shared memory traffic will be published to **NDDS_Transport_Shmem_Property_t::udp_debugging_address** (p. 2054) :: **NDDS_Transport_Shmem_Property_t::udp_debugging_port** (p. 2055).

[default] 0

9.340.2.5 udp_debugging_address

NDDS_Transport_Address_t NDDS_Transport_Shmem_Property_t::udp_debugging_address

IP address to which shared memory traffic will be published if **NDDS_Transport_Shmem_Property_t::enable_udp_debugging** (p. 2054) is set to '1'.

[default] 239.255.1.2

9.340.2.6 udp_debugging_port

NDDS_Transport_Port_t NDDS_Transport_Shmem_Property_t::udp_debugging_port

Port to which shared memory traffic will be published if **NDDS_Transport_Shmem_Property_t::enable_udp_debugging** (p. 2054) is set to '1'.

[default] 7399

9.341 NDDS_Transport_UDP_WAN_CommPortsMappingInfo Struct Reference

Type for storing UDP WAN communication ports.

`#include <transport_udp.h>`

Public Attributes

- **NDDS_Transport_Port_t rtps_port**
  RTPS port.
- **NDDS_Transport_UDP_Port host_port**
  Host port.
- **NDDS_Transport_UDP_Port public_port**
  Public port.

9.341.1 Detailed Description

Type for storing UDP WAN communication ports.

9.341.2 Member Data Documentation

9.341.2.1 rtps_port

NDDS_Transport_Port_t NDDS_Transport_UDP_WAN_CommPortsMappingInfo::rtps_port

RTPS port.

9.341.2.2 host_port

NDDS_Transport_UDP_Port NDDS_Transport_UDP_WAN_CommPortsMappingInfo::host_port

Host port.

9.341.2.3 public_port

NDDS_Transport_UDP_Port NDDS_Transport_UDP_WAN_CommPortsMappingInfo::public_port

Public port.

9.342 NDDS_Transport_UDPv4_Property_t Struct Reference

Configurable IPv4/UDP Transport-Plugin properties.

`#include <transport_udpv4.h>`

Public Attributes

- struct **NDDS_Transport_Property_t parent**
  Generic properties of all transport plugins.
- RTI_INT32 **send_socket_buffer_size**
  Size in bytes of the send buffer of a socket used for sending.
- RTI_INT32 **recv_socket_buffer_size**
  Size in bytes of the receive buffer of a socket used for receiving.
- RTI_INT32 **unicast_enabled**
  Allows the transport plugin to use unicast for sending and receiving.
- RTI_INT32 **multicast_enabled**
  Allows the transport plugin to use multicast for sending and receiving.
- RTI_INT32 **multicast_ttl**
  Value for the time-to-live parameter for all multicast sends using this plugin.
- RTI_INT32 **multicast_loopback_disabled**
  Prevents the transport plugin from putting multicast packets onto the loopback interface.

- RTI_INT32 **ignore_loopback_interface**

  Prevents the transport plugin from using the IP loopback interface.
- RTI_INT32 **ignore_nonup_interfaces**

  [DEPRECATED] *Prevents the transport plugin from using a network interface that is not reported as UP by the operating system.*
- RTI_INT32 **ignore_nonrunning_interfaces**

  Prevents the transport plugin from using a network interface that is not reported as RUNNING by the operating system.
- RTI_INT32 **no_zero_copy**

  [DEPRECATED] *Prevents the transport plugin from doing a zero copy.*
- RTI_INT32 **send_blocking**

  Control blocking behavior of send sockets. CHANGING THIS FROM THE DEFAULT CAN CAUSE SIGNIFICANT PERFORMANCE PROBLEMS.
- RTI_INT32 **use_checksum**

  Configures whether to send UDP checksum.
- RTI_UINT32 **transport_priority_mask**

  Set mask for use of transport priority field.
- RTI_INT32 **transport_priority_mapping_low**

  Set low value of output range to IPv4 TOS.
- RTI_INT32 **transport_priority_mapping_high**

  Set high value of output range to IPv4 TOS.
- RTI_INT32 **send_ping**

  Configures whether to send PING messages.
- RTI_INT32 **force_interface_poll_detection**

  Forces the interface tracker to use a polling mechanism to detect changes on the UDPv4 interfaces.
- RTI_UINT32 **interface_poll_period**

  Specifies the period in milliseconds to query for changes in the state of all the interfaces.
- RTI_INT32 **reuse_multicast_receive_resource**

  Controls whether or not to reuse multicast receive resources.
- RTI_INT32 **protocol_overhead_max**

  Maximum size in bytes of protocol overhead, including headers.
- RTI_INT32 **disable_interface_tracking**

  Disables detection of network interface changes.
- RTI_UINT32 **join_multicast_group_timeout**

  [Windows only] Defines how much time (milliseconds) to wait to join a multicast group address when a new interface is detected.
- char * **public_address**

  Public IP address associated with the transport instantiation.
- RTI_INT32 **port_offset**

  Port offset to allow multiple instances of UDPv4 transports in the same ***DDSDomainParticipant*** *(p. 1619).*

9.342.1 Detailed Description

Configurable IPv4/UDP Transport-Plugin properties.

You can modify the properties in this structure to configure the plugin. However, you must set the properties before the plugin is instantiated.

See also

NDDSTransportSupport::set_builtin_transport_property() (p. 2109)

NDDS_Transport_UDPv4_new (p. 470)

9.342.2 Member Data Documentation

9.342.2.1 parent

`struct` **NDDS_Transport_Property_t** NDDS_Transport_UDPv4_Property_t::parent

Generic properties of all transport plugins.

9.342.2.2 send_socket_buffer_size

RTI_INT32 NDDS_Transport_UDPv4_Property_t::send_socket_buffer_size

Size in bytes of the send buffer of a socket used for sending.

On most operating systems, `setsockopt()` will be called to set the SENDBUF to the value of this parameter.

This value must be greater than or equal to **NDDS_Transport_Property_t::message_size_max** (p. 2047). Its maximum value is dependent on the operating system.

By default, it will be set to **NDDS_TRANSPORT_UDPV4_SEND_SOCKET_BUFFER_SIZE_DEFAULT** (p. 468).

The maximum value for this property depends on the operating system.

Valid values are:

- **NDDS_TRANSPORT_UDPV4_SEND_SOCKET_BUFFER_SIZE_DEFAULT** (p. 468): `setsockopt()` (or equivalent) will be called to size the send buffer of the socket with the default value.
- **NDDS_TRANSPORT_UDPV4_SOCKET_BUFFER_SIZE_OS_DEFAULT** (p. 468): `setsockopt()` (or equivalent) will not be called to size the send buffer of the socket. RTI Connext will use the OS default.
- **NDDS_TRANSPORT_UDPV4_SOCKET_BUFFER_SIZE_OS_MAX** (p. 467): RTI Connext will retrieve the maximum socket buffer size allowed by the operating system and use that value for the send buffer size of the socket.
- Positive integer: `setsockopt()` (or equivalent) will be called to size the send buffer of the socket with the provided value. If this value exceeds the maximum allowed by the operating system, the operating system's maximum value will be used instead.

9.342.2.3 recv_socket_buffer_size

```
RTI_INT32 NDDS_Transport_UDPv4_Property_t::recv_socket_buffer_size
```

Size in bytes of the receive buffer of a socket used for receiving.

On most operating systems, `setsockopt()` will be called to set the RECVBUF to the value of this parameter.

This value must be greater than or equal to **NDDS_Transport_Property_t::message_size_max** (p. 2047). Its maximum value is dependent on the operating system.

By default, it will be set to **NDDS_TRANSPORT_UDPV4_RECV_SOCKET_BUFFER_SIZE_DEFAULT** (p. 468).

The maximum value for this property depends on the operating system.

Valid values are:

- **NDDS_TRANSPORT_UDPV4_RECV_SOCKET_BUFFER_SIZE_DEFAULT** (p. 468): `setsockopt()` (or equivalent) will be called to size the receive buffer of the socket with the default value.
- **NDDS_TRANSPORT_UDPV4_SOCKET_BUFFER_SIZE_OS_DEFAULT** (p. 468): `setsockopt()` (or equivalent) will not be called to size the receive buffer of the socket. RTI Connext will use the OS default.
- **NDDS_TRANSPORT_UDPV4_SOCKET_BUFFER_SIZE_OS_MAX** (p. 467): RTI Connext will retrieve the maximum socket buffer size allowed by the operating system and use that value for the receive buffer size of the socket.
- Positive integer: `setsockopt()` (or equivalent) will be called to size the receive buffer of the socket with the provided value. If this value exceeds the maximum allowed by the operating system, the operating system's maximum value will be used instead.

9.342.2.4 unicast_enabled

```
RTI_INT32 NDDS_Transport_UDPv4_Property_t::unicast_enabled
```

Allows the transport plugin to use unicast for sending and receiving.

This value turns unicast UDP on (if set to 1) or off (if set to 0) for this plugin. By default, it will be turned on (1). Also by default, the plugin will use all the allowed network interfaces that it finds up and running when the plugin is instantiated.

9.342.2.5 multicast_enabled

```
RTI_INT32 NDDS_Transport_UDPv4_Property_t::multicast_enabled
```

Allows the transport plugin to use multicast for sending and receiving.

This value turns multicast UDP on (if set to 1) or off (if set to 0) for this plugin. By default, it will be turned on (1) for those platforms that support multicast. Also by default, the plugin will use all the network interfaces allowed for multicast that it finds up and running when the plugin is instantiated.

9.342.2.6 multicast_ttl

```
RTI_INT32 NDDS_Transport_UDPv4_Property_t::multicast_ttl
```

Value for the time-to-live parameter for all multicast sends using this plugin.

This value is used to set the TTL of multicast packets sent by this transport plugin.

[default] 1

See also

NDDS_TRANSPORT_UDPV4_MULTICAST_TTL_DEFAULT (p. 469)

9.342.2.7 multicast_loopback_disabled

```
RTI_INT32 NDDS_Transport_UDPv4_Property_t::multicast_loopback_disabled
```

Prevents the transport plugin from putting multicast packets onto the loopback interface.

If multicast loopback is disabled (this value is set to 1), then when sending multicast packets, RTI Connext will *not* put a copy of the packets on the loopback interface. This prevents applications on the same node (including itself) from receiving those packets.

This value is set to 0 by default, meaning multicast loopback is *enabled*.

Disabling multicast loopback (setting this value to 1) may result in minor performance gains when using multicast.

[NOTE: Windows CE systems do not support multicast loopback. This field is ignored for Windows CE targets.]

9.342.2.8 ignore_loopback_interface

```
RTI_INT32 NDDS_Transport_UDPv4_Property_t::ignore_loopback_interface
```

Prevents the transport plugin from using the IP loopback interface.

Currently three values are allowed:

- **0**: Forces local traffic to be sent over loopback, even if a more efficient transport (such as shared memory) is installed (in which case traffic will be sent over both transports).
- **1**: Disables local traffic via this plugin. The IP loopback interface is not used, even if no NICs are discovered. This is useful when you want applications running on the same node to use a more efficient plugin (such as shared memory) instead of the IP loopback.
- **-1**: Automatic. Lets RTI Connext decide between the above two choices.

The current "automatic" (-1) RTI Connext policy is as follows:

- If a shared memory transport plugin is available for local traffic and there is a locator on the initial peers list that can use shared memory, the effective value is 1 (i.e., disable UPV4 local traffic).
- Otherwise, the effective value is 0 (i.e., use UDPv4 for local traffic also).

[default] -1 Automatic RTI Connext policy based on availability of the shared memory transport.

9.342.2.9 ignore_nonup_interfaces

```
RTI_INT32 NDDS_Transport_UDPv4_Property_t::ignore_nonup_interfaces
```

[DEPRECATED] Prevents the transport plugin from using a network interface that is not reported as UP by the operating system.

DEPRECATED: this property has no effect. Non-UP interfaces are ignored until they change their status to UP, unless **NDDS_Transport_UDPv4_Property_t::disable_interface_tracking** (p. 2065) is set to 1, in which case interfaces are ignored even if they change their status at a later point.

The transport checks the flags reported by the operating system for each network interface upon initialization. An interface that is not reported as UP will not be used. This property allows you to configure the transport to start using even the interfaces that were not reported as UP.

Two values are allowed:

- **0**: Allow the use of interfaces that were not reported as UP.
- **1**: Do not use interfaces that were not reported as UP.

[default] 1

9.342.2.10 ignore_nonrunning_interfaces

```
RTI_INT32 NDDS_Transport_UDPv4_Property_t::ignore_nonrunning_interfaces
```

Prevents the transport plugin from using a network interface that is not reported as RUNNING by the operating system.

The transport checks the flags reported by the operating system for each network interface upon initialization. An interface that is not reported as UP will not be used. This property allows the same check to be extended to the IFF_↩ RUNNING flag implemented by some operating systems. The RUNNING flag is defined to mean that "all resources are allocated," and may be off if there is no link detected, e.g., the network cable is unplugged.

Two values are allowed:

- **0**: Do not check the RUNNING flag when enumerating interfaces, just make sure the interface is UP.
- **1**: Check the flag when enumerating interfaces, and ignore those that are not reported as RUNNING. This can be used on some operating systems to cause the transport to ignore interfaces that are enabled but not connected to the network.

[default] 1 (i.e., check RUNNING flag)

9.342.2.11 no_zero_copy

```
RTI_INT32 NDDS_Transport_UDPv4_Property_t::no_zero_copy
```

[DEPRECATED] Prevents the transport plugin from doing a zero copy.

DEPRECATED: This property has no effect. By default, this plugin will use the zero copy on OSes that offer it. While this is good for performance, it may sometimes tax the OS resources in a manner that cannot be overcome by the application.

The best example is if the hardware/device driver lends the buffer to the application itself. If the application does not return the loaned buffers soon enough, the node may err or malfunction. In case you cannot reconfigure the H/W, device driver, or the OS to allow the zero copy feature to work for your application, you may have no choice but to turn off zero copy use.

By default this is set to 0, so RTI Connext will use the zero-copy API if offered by the OS.

9.342.2.12 send_blocking

```
RTI_INT32 NDDS_Transport_UDPv4_Property_t::send_blocking
```

Control blocking behavior of send sockets. CHANGING THIS FROM THE DEFAULT CAN CAUSE SIGNIFICANT PERFORMANCE PROBLEMS.

Currently two values are defined:

- **NDDS_TRANSPORT_UDPV4_BLOCKING_ALWAYS:** Sockets are blocking (default socket options for the operating system).
- **NDDS_TRANSPORT_UDPV4_BLOCKING_NEVER:** Sockets are modified to make them non-blocking. THIS MAY CAUSE SIGNIFICANT PERFORMANCE PROBLEMS.

[default] NDDS_TRANSPORT_UDPV4_BLOCKING_ALWAYS

9.342.2.13 use_checksum

```
RTI_INT32 NDDS_Transport_UDPv4_Property_t::use_checksum
```

Configures whether to send UDP checksum.

This property specifies whether the UDP checksum will be computed. On Windows and Linux, the UDP protocol will not set the checksum when use_checksum is set to 0. This is useful when RTPS protocol statistics related to corrupted messages need to be collected through the API **DDSDomainParticipant::get_participant_protocol_status** (p. 1676).

[default] 1 (enabled)

9.342.2.14 transport_priority_mask

```
RTI_UINT32 NDDS_Transport_UDPv4_Property_t::transport_priority_mask
```

Set mask for use of transport priority field.

This is used in conjunction with **NDDS_Transport_UDPv4_Property_t::transport_priority_mapping_low** (p. 2063) and **NDDS_Transport_UDPv4_Property_t::transport_priority_mapping_high** (p. 2063) to define the mapping from the DDS transport priority (see **TRANSPORT_PRIORITY** (p. 647)) to the IPv4 TOS field. Defines a contiguous region of bits in the 32-bit transport priority value that is used to generate values for the IPv4 TOS field on an outgoing socket.

For example, the value 0x0000ff00 causes bits 9-16 (8 bits) to be used in the mapping. The value will be scaled from the mask range (0x0000 - 0xff00 in this case) to the range specified by low and high.

If the mask is set to zero, then the transport will not set IPv4 TOS for send sockets.

[default] 0.

9.342.2.15 transport_priority_mapping_low

```
RTI_INT32 NDDS_Transport_UDPv4_Property_t::transport_priority_mapping_low
```

Set low value of output range to IPv4 TOS.

This is used in conjunction with **NDDS_Transport_UDPv4_Property_t::transport_priority_mask** (p. 2062) and **NDDS_Transport_UDPv4_Property_t::transport_priority_mapping_high** (p. 2063) to define the mapping from the DDS transport priority to the IPv4 TOS field. Defines the low value of the output range for scaling.

Note that IPv4 TOS is generally an 8-bit value.

[default] 0.

9.342.2.16 transport_priority_mapping_high

```
RTI_INT32 NDDS_Transport_UDPv4_Property_t::transport_priority_mapping_high
```

Set high value of output range to IPv4 TOS.

This is used in conjunction with **NDDS_Transport_UDPv4_Property_t::transport_priority_mask** (p. 2062) and **NDDS_Transport_UDPv4_Property_t::transport_priority_mapping_low** (p. 2063) to define the mapping from the DDS transport priority to the IPv4 TOS field. Defines the high value of the output range for scaling.

Note that IPv4 TOS is generally an 8-bit value.

[default] 0xff.

9.342.2.17 send_ping

```
RTI_INT32 NDDS_Transport_UDPv4_Property_t::send_ping
```

Configures whether to send PING messages.

This property specifies whether to send a PING message before commencing the discovery process. On certain operating systems or with certain switches the initial UDP packet, while configuring the ARP table, can unfortunately be dropped. To avoid dropping the initial RTPS discovery sample, a PING message is sent to preconfigure the ARP table in those environments.

[default] 1 (enabled)

9.342.2.18 force_interface_poll_detection

```
RTI_INT32 NDDS_Transport_UDPv4_Property_t::force_interface_poll_detection
```

Forces the interface tracker to use a polling mechanism to detect changes on the UDPv4 interfaces.

When possible, the detection of an IP address change is done asynchronously using the APIs offered by the underlying OS. By setting this property on those OSes, the use of a polling mechanism to detect changes can be forced.

[default] 0 (disabled).

9.342.2.19 interface_poll_period

```
RTI_UINT32 NDDS_Transport_UDPv4_Property_t::interface_poll_period
```

Specifies the period in milliseconds to query for changes in the state of all the interfaces.

When possible, the detection of an IP address change is done asynchronously using the APIs offered by the underlying OS. If there is no mechanism to do that, the detection will use a polling strategy where the polling period can be configured by setting this property.

[default] 500 milliseconds.

9.342.2.20 reuse_multicast_receive_resource

```
RTI_INT32 NDDS_Transport_UDPv4_Property_t::reuse_multicast_receive_resource
```

Controls whether or not to reuse multicast receive resources.

Setting this to 0 (FALSE) prevents multicast crosstalk by uniquely configuring a port and creating a receive thread for each multicast group address.

[default] 1.

9.342.2.21 protocol_overhead_max

```
RTI_INT32 NDDS_Transport_UDPv4_Property_t::protocol_overhead_max
```

Maximum size in bytes of protocol overhead, including headers.

This value is the maximum size, in bytes, of protocol-related overhead. Normally, the overhead accounts for UDP and IP headers. The default value is set to accommodate the most common UDP/IP header size.

Note that when **NDDS_Transport_Property_t::message_size_max** (p. 2047) plus this overhead is larger than the UDPv4 maximum message size (65535 bytes), the middleware will automatically reduce the effective message_size↩ _max, to 65535 minus this overhead.

[default] 28.

See also

NDDS_Transport_Property_t::message_size_max (p. 2047)

9.342.2.22 disable_interface_tracking

```
RTI_INT32 NDDS_Transport_UDPv4_Property_t::disable_interface_tracking
```

Disables detection of network interface changes.

By default, network interface changes are propagated in the form of locators to other applications. This is done to support IP mobility scenarios.

For example, you could start a RTI Connext application with Wi-Fi and move to a wired connection. In order to continue communicating with other applications, this interface change has to be propagated.

In RTI Connext 5.2 (the initial release) and earlier versions of the product, IP mobility scenarios were not supported. 5.2 applications will report errors if they detect locator changes in a DataWriter or DataReader.

You can disable the notification and propagation of interface changes by setting this property to 1.

This way, an interface change in a newer application will not trigger errors in an application running 5.2 or earlier. Of course, this will prevent the new application from being able to detect network interface changes.

[default] 0

9.342.2.23 join_multicast_group_timeout

```
RTI_UINT32 NDDS_Transport_UDPv4_Property_t::join_multicast_group_timeout
```

[Windows only] Defines how much time (milliseconds) to wait to join a multicast group address when a new interface is detected.

On Windows, a network interface may be detected before it is allowed to join a multicast group address. This property adjusts how much time (milliseconds) to wait for the ADD_MEMBERSHIP multicast operation to succeed before withdrawing.

[default] 5000

9.342.2.24 public_address

```
char* NDDS_Transport_UDPv4_Property_t::public_address
```

Public IP address associated with the transport instantiation.

Setting the public IP address is only necessary to support communication over WAN that involves Network Address Translation (NAT).

Typically, the address is the public address of the IP NAT router that provides access to the WAN.

By default, the **DDSDomainParticipant** (p. 1619) creating the transport will announce the IP addresses obtained from the NICs to other DomainParticipants in the system.

When this property is set, the DomainParticipant will announce the IP address corresponding to the property value instead of the LAN IP addresses associated with the NICs.

Note 1: Setting this property is necessary, but is not a sufficient condition for sending and receiving data over the WAN. You must also configure the IP NAT router to allow UDP traffic and to map the public IP address specified by this property to the DomainParticipant's private LAN IP address. This is typically done with one of the following mechanisms:

- Port Forwarding: You must map the private ports used to receive discovery and user data traffic to the corresponding public ports (see **DDS_RtpsWellKnownPorts_t** (p. 1338)). Public and private ports must be the same since the transport does not allow you to change the mapping.
- 1:1 NAT: You must add a 1:1 NAT entry that maps the public IP address specified in this property to the private LAN IP address of the DomainParticipant.

Note 2: By setting this property, the **DDSDomainParticipant** (p. 1619) only announces its public IP address to other DomainParticipants. Therefore, communication with DomainParticipants within the LAN that are running on different nodes will not work unless the NAT router is configured to enable NAT reflection (hairpin NAT).

There is another way to achieve simultaneous communication with DomainParticipants running in the LAN and WAN, that does not require hairpin NAT. This way uses a gateway application such as RTI Routing Service to provide access to the WAN.

[default] NULL (the transport uses the IP addresses obtained from the NICs)

9.342.2.25 port_offset

```
RTI_INT32 NDDS_Transport_UDPv4_Property_t::port_offset
```

Port offset to allow multiple instances of UDPv4 transports in the same **DDSDomainParticipant** (p. 1619).

By default, it is not possible to have multiple instances of the UDPv4 transport in the same **DDSDomainParticipant** (p. 1619). This is because all instances will try to bind to the same UDP port(s) for receiving data.

With this property, you can specify an offset that will be added to the RTPS port(s) to determine the UDP port(s) to bind to. This way, you can have multiple instances of the UDPv4 transport in the same **DDSDomainParticipant** (p. 1619).

An RTPS port will be mapped to an UDP port as follows: UDP port = RTPS port + port_offset.

[default] 0

9.343 NDDS_Transport_UDPv4_WAN_Property_t Struct Reference

Configurable IPv4/UDP WAN Transport-Plugin properties.

`#include <transport_udpv4.h>`

Public Attributes

- struct **NDDS_Transport_Property_t parent**

  Generic properties of all transport plugins.
- RTI_INT32 **send_socket_buffer_size**

  Size in bytes of the send buffer of a socket used for sending.
- RTI_INT32 **recv_socket_buffer_size**

  Size in bytes of the receive buffer of a socket used for receiving.
- RTI_INT32 **ignore_loopback_interface**

  Prevents the transport plugin from using the IP loopback interface.
- RTI_INT32 **ignore_nonup_interfaces**

  ***[DEPRECATED]** Prevents the transport plugin from using a network interface that is not reported as UP by the operating system.*
- RTI_INT32 **ignore_nonrunning_interfaces**

  Prevents the transport plugin from using a network interface that is not reported as RUNNING by the operating system.
- RTI_INT32 **no_zero_copy**

  ***[DEPRECATED]** Prevents the transport plugin from doing a zero copy.*
- RTI_INT32 **send_blocking**

  Control blocking behavior of send sockets. CHANGING THIS FROM THE DEFAULT CAN CAUSE SIGNIFICANT PERFORMANCE PROBLEMS.
- RTI_INT32 **use_checksum**

  Configures whether to send UDP checksum.
- RTI_UINT32 **transport_priority_mask**

  Set mask for use of transport priority field.
- RTI_INT32 **transport_priority_mapping_low**

  Set low value of output range to IPv4 TOS.
- RTI_INT32 **transport_priority_mapping_high**

  Set high value of output range to IPv4 TOS.
- RTI_INT32 **send_ping**

  Configures whether to send PING messages.
- RTI_INT32 **force_interface_poll_detection**

  Forces the interface tracker to use a polling mechanism to detect changes on the UDPv4 interfaces.
- RTI_UINT32 **interface_poll_period**

  Specifies the period in milliseconds to query for changes in the state of all the interfaces.
- RTI_INT32 **protocol_overhead_max**

  Maximum size in bytes of protocol overhead, including headers.
- RTI_INT32 **disable_interface_tracking**

  Disables detection of network interface changes.
- char * **public_address**

  Public IP address associated with the transport instantiation.
- struct **NDDS_Transport_UDP_WAN_CommPortsMappingInfo** * **comm_ports_list**

Configures the public and private UDP ports that a transport instance uses to receive/send RTPS data.

- RTI_INT32 **comm_ports_list_length**

  Number of elements in the ***NDDS_Transport_UDPv4_WAN_Property_t::comm_ports_list*** *(p. 2074).*

- RTI_INT32 **port_offset**

  Port offset to allow coexistence with built-in UDPv4 transport.

- RTI_UINT32 **binding_ping_period**

  Specifies the period in milliseconds at which BINDING PINGS messages are sent to keep NAT mappings open.

9.343.1 Detailed Description

Configurable IPv4/UDP WAN Transport-Plugin properties.

You can modify the properties in this structure to configure the plugin. However, you must set the properties before the plugin is instantiated.

See also

NDDSTransportSupport::set_builtin_transport_property() (p. 2109)

NDDS_Transport_UDPv4_WAN_new (p. 477)

9.343.2 Member Data Documentation

9.343.2.1 parent

```
struct NDDS_Transport_Property_t NDDS_Transport_UDPv4_WAN_Property_t::parent
```

Generic properties of all transport plugins.

9.343.2.2 send_socket_buffer_size

```
RTI_INT32 NDDS_Transport_UDPv4_WAN_Property_t::send_socket_buffer_size
```

Size in bytes of the send buffer of a socket used for sending.

See also

NDDS_Transport_UDPv4_Property_t::send_socket_buffer_size (p. 2058)

9.343.2.3 recv_socket_buffer_size

```
RTI_INT32 NDDS_Transport_UDPv4_WAN_Property_t::recv_socket_buffer_size
```

Size in bytes of the receive buffer of a socket used for receiving.

See also

NDDS_Transport_UDPv4_Property_t::recv_socket_buffer_size (p. 2058)

9.343.2.4 ignore_loopback_interface

```
RTI_INT32 NDDS_Transport_UDPv4_WAN_Property_t::ignore_loopback_interface
```

Prevents the transport plugin from using the IP loopback interface.

Currently three values are allowed:

- **0**: Forces local traffic to be sent over loopback, even if a more efficient transport (such as shared memory) is installed (in which case traffic will be sent over both transports).
- **1**: Disables local traffic via this plugin. The IP loopback interface is not used, even if no NICs are discovered. This is useful when you want applications running on the same node to use a more efficient plugin (such as shared memory) instead of the IP loopback.
- **-1**: Automatic. Lets RTI Connext decide between the above two choices.

The current "automatic" (-1) RTI Connext policy is as follows:

- If a shared memory transport plugin is available for local traffic and there is a locator on the initial peers list that can use shared memory, the effective value is 1 (i.e., disable UPV4 local traffic).
- Otherwise, the effective value is 0 (i.e., use UDPv4 for local traffic also).

[default] -1 Automatic RTI Connext policy based on availability of the shared memory transport.

9.343.2.5 ignore_nonup_interfaces

```
RTI_INT32 NDDS_Transport_UDPv4_WAN_Property_t::ignore_nonup_interfaces
```

[DEPRECATED] Prevents the transport plugin from using a network interface that is not reported as UP by the operating system.

DEPRECATED: this property has no effect. Non-UP interfaces are ignored until they change their status to UP, unless **NDDS_Transport_UDPv4_WAN_Property_t::disable_interface_tracking** (p. 2073) is set to 1, in which case interfaces are ignored even if they change their status at a later point.

The transport checks the flags reported by the operating system for each network interface upon initialization. An interface that is not reported as UP will not be used. This property allows you to configure the transport to start using even the interfaces that were not reported as UP.

Two values are allowed:

- **0**: Allow the use of interfaces that were not reported as UP.
- **1**: Do not use interfaces that were not reported as UP.

[default] 1

9.343.2.6 ignore_nonrunning_interfaces

```
RTI_INT32 NDDS_Transport_UDPv4_WAN_Property_t::ignore_nonrunning_interfaces
```

Prevents the transport plugin from using a network interface that is not reported as RUNNING by the operating system.

The transport checks the flags reported by the operating system for each network interface upon initialization. An interface that is not reported as UP will not be used. This property allows the same check to be extended to the IFF_↩ RUNNING flag implemented by some operating systems. The RUNNING flag is defined to mean that "all resources are allocated," and may be off if there is no link detected, e.g., the network cable is unplugged.

Two values are allowed:

- **0**: Do not check the RUNNING flag when enumerating interfaces, just make sure the interface is UP.
- **1**: Check the flag when enumerating interfaces, and ignore those that are not reported as RUNNING. This can be used on some operating systems to cause the transport to ignore interfaces that are enabled but not connected to the network.

[default] 1 (i.e., check RUNNING flag)

9.343.2.7 no_zero_copy

```
RTI_INT32 NDDS_Transport_UDPv4_WAN_Property_t::no_zero_copy
```

[DEPRECATED] Prevents the transport plugin from doing a zero copy.

DEPRECATED: This property has no effect. By default, this plugin will use the zero copy on OSes that offer it. While this is good for performance, it may sometimes tax the OS resources in a manner that cannot be overcome by the application.

The best example is if the hardware/device driver lends the buffer to the application itself. If the application does not return the loaned buffers soon enough, the node may err or malfunction. In case you cannot reconfigure the H/W, device driver, or the OS to allow the zero copy feature to work for your application, you may have no choice but to turn off zero copy use.

By default this is set to 0, so RTI Connext will use the zero-copy API if offered by the OS.

9.343.2.8 send_blocking

```
RTI_INT32 NDDS_Transport_UDPv4_WAN_Property_t::send_blocking
```

Control blocking behavior of send sockets. CHANGING THIS FROM THE DEFAULT CAN CAUSE SIGNIFICANT PERFORMANCE PROBLEMS.

Currently two values are defined:

- **NDDS_TRANSPORT_UDPV4_BLOCKING_ALWAYS:** Sockets are blocking (default socket options for the operating system).
- **NDDS_TRANSPORT_UDPV4_BLOCKING_NEVER:** Sockets are modified to make them non-blocking. THIS MAY CAUSE SIGNIFICANT PERFORMANCE PROBLEMS.

[default] NDDS_TRANSPORT_UDPV4_BLOCKING_ALWAYS

This property is not supported on Windows when using the Real-Time WAN Transport. The send sockets cannot be configured to be non-blocking because Windows does not provide a way to configure a socket to only be non-blocking for send operations but not receive operations.

9.343.2.9 use_checksum

```
RTI_INT32 NDDS_Transport_UDPv4_WAN_Property_t::use_checksum
```

Configures whether to send UDP checksum.

This property specifies whether the UDP checksum will be computed. On Windows and Linux, the UDP protocol will not set the checksum when use_checksum is set to 0. This is useful when RTPS protocol statistics related to corrupted messages need to be collected through the API **DDSDomainParticipant::get_participant_protocol_status** (p. 1676).

[default] 1 (enabled)

9.343.2.10 transport_priority_mask

```
RTI_UINT32 NDDS_Transport_UDPv4_WAN_Property_t::transport_priority_mask
```

Set mask for use of transport priority field.

This is used in conjunction with **NDDS_Transport_UDPv4_WAN_Property_t::transport_priority_mapping_low** (p. 2072) and **NDDS_Transport_UDPv4_WAN_Property_t::transport_priority_mapping_high** (p. 2072) to define the mapping from the DDS transport priority (see **TRANSPORT_PRIORITY** (p. 647)) to the IPv4 TOS field. Defines a contiguous region of bits in the 32-bit transport priority value that is used to generate values for the IPv4 TOS field on an outgoing socket.

For example, the value 0x0000ff00 causes bits 9-16 (8 bits) to be used in the mapping. The value will be scaled from the mask range (0x0000 - 0xff00 in this case) to the range specified by low and high.

If the mask is set to zero, then the transport will not set IPv4 TOS for send sockets.

[default] 0.

9.343.2.11 transport_priority_mapping_low

```
RTI_INT32 NDDS_Transport_UDPv4_WAN_Property_t::transport_priority_mapping_low
```

Set low value of output range to IPv4 TOS.

This is used in conjunction with **NDDS_Transport_UDPv4_WAN_Property_t::transport_priority_mask** (p. 2071) and **NDDS_Transport_UDPv4_WAN_Property_t::transport_priority_mapping_high** (p. 2072) to define the mapping from the DDS transport priority to the IPv4 TOS field. Defines the low value of the output range for scaling.

Note that IPv4 TOS is generally an 8-bit value.

[default] 0.

9.343.2.12 transport_priority_mapping_high

```
RTI_INT32 NDDS_Transport_UDPv4_WAN_Property_t::transport_priority_mapping_high
```

Set high value of output range to IPv4 TOS.

This is used in conjunction with **NDDS_Transport_UDPv4_WAN_Property_t::transport_priority_mask** (p. 2071) and **NDDS_Transport_UDPv4_WAN_Property_t::transport_priority_mapping_low** (p. 2072) to define the mapping from the DDS transport priority to the IPv4 TOS field. Defines the high value of the output range for scaling.

Note that IPv4 TOS is generally an 8-bit value.

[default] 0xff.

9.343.2.13 send_ping

```
RTI_INT32 NDDS_Transport_UDPv4_WAN_Property_t::send_ping
```

Configures whether to send PING messages.

This property specifies whether to send a PING message before commencing the discovery process. On certain operating systems or with certain switches the initial UDP packet, while configuring the ARP table, can unfortunately be dropped. To avoid dropping the initial RTPS discovery sample, a PING message is sent to preconfigure the ARP table in those environments.

[default] 1 (enabled)

9.343.2.14 force_interface_poll_detection

```
RTI_INT32 NDDS_Transport_UDPv4_WAN_Property_t::force_interface_poll_detection
```

Forces the interface tracker to use a polling mechanism to detect changes on the UDPv4 interfaces.

When possible, the detection of an IP address change is done asynchronously using the APIs offered by the underlying OS. By setting this property on those OSes, the use of a polling mechanism to detect changes can be forced.

[default] 0 (disabled).

9.343.2.15 interface_poll_period

```
RTI_UINT32 NDDS_Transport_UDPv4_WAN_Property_t::interface_poll_period
```

Specifies the period in milliseconds to query for changes in the state of all the interfaces.

When possible, the detection of an IP address change is done asynchronously using the APIs offered by the underlying OS. If there is no mechanism to do that, the detection will use a polling strategy where the polling period can be configured by setting this property.

[default] 500 milliseconds.

9.343.2.16 protocol_overhead_max

```
RTI_INT32 NDDS_Transport_UDPv4_WAN_Property_t::protocol_overhead_max
```

Maximum size in bytes of protocol overhead, including headers.

This value is the maximum size, in bytes, of protocol-related overhead. Normally, the overhead accounts for UDP and IP headers. The default value is set to accommodate the most common UDP/IP header size.

Note that when **NDDS_Transport_Property_t::message_size_max** (p. 2047) plus this overhead is larger than the UDPv4 maximum message size (65535 bytes), the middleware will automatically reduce the effective message_size↩ _max, to 65535 minus this overhead.

[default] 28.

See also

NDDS_Transport_Property_t::message_size_max (p. 2047)

9.343.2.17 disable_interface_tracking

```
RTI_INT32 NDDS_Transport_UDPv4_WAN_Property_t::disable_interface_tracking
```

Disables detection of network interface changes.

By default, network interface changes are propagated in the form of locators to other applications. This is done to support IP mobility scenarios.

For example, you could start a RTI Connext application with Wi-Fi and move to a wired connection. In order to continue communicating with other applications, this interface change has to be propagated.

In RTI Connext 5.2 (the initial release) and earlier versions of the product, IP mobility scenarios were not supported. 5.2 applications will report errors if they detect locator changes in a DataWriter or DataReader.

You can disable the notification and propagation of interface changes by setting this property to 1.

This way, an interface change in a newer application will not trigger errors in an application running 5.2 or earlier. Of course, this will prevent the new application from being able to detect network interface changes.

[default] 0

9.343.2.18 public_address

```
char* NDDS_Transport_UDPv4_WAN_Property_t::public_address
```

Public IP address associated with the transport instantiation.

Setting the public IP address is only necessary for the Real-Time WAN Transport associated with an external **DDSDomainParticipant** (p. 1619) (publicly reachable) in order to support the two communication scenarios shown in the Figure below.

For an external **DDSDomainParticipant** (p. 1619) behind a NAT-enabled router, this address is the public IP address of the router.

When this property is set, the DomainParticipant will announce PUBLIC+UUID locators to other DomainParticipants. These locators are reachable locators because they contain this public IP address.

For additional information on Real-Time WAN Transport locators, see the `Core Libraries User's Manual`.

with a Participant that has a Public Address"

[default] NULL (the transport will announce UUID locators)

9.343.2.19 comm_ports_list

struct **NDDS_Transport_UDP_WAN_CommPortsMappingInfo*** NDDS_Transport_UDPv4_WAN_Property_t::comm_↩ports_list

Configures the public and private UDP ports that a transport instance uses to receive/send RTPS data.

Array containing the mapping between "RTPS ports", "UDP receive host ports", and "UDP receive public ports".

When the transport is configured using properties, the port mapping array is provided using a JSON string.

For example:

```
{
    "default": {"host": 8192, "public": 9678},
    "mappings": [
        {"rtps": 1234, "host": 9999, "public": 5678},
        {"rtps": 1235, "host": 9990, "public": 5679},
    ]
}
```

It is also possible to configure the mapping with XML:

```
<transport_builtin>
    <udpv4_wan>
        <comm_ports>
            <default>
                <host>8192</host>
                <public>9678</public>
            </default>
            <mappings>
                <element>
                    <rtps>1234</rtps>
                    <host>9999</host>
                    <public>5678</public>
                </element>
                <element>
                    <rtps>1235</rtps>
                    <host>9990</host>
                    <public>5679</public>
                </element>
            </mappings>
        </comm_ports>
    </udpv4_wan>
<transport_builtin>
```

For additional information on how to set the value of this property, see the `Core Libraries User's Manual`.

[default] NULL (The UDP ports used for communications will be derived from the RTPS ports associated with the locators for the DomainParticipant and its Endpoints (DataWriters and DataReaders)).

9.343.2.20 comm_ports_list_length

```
RTI_INT32 NDDS_Transport_UDPv4_WAN_Property_t::comm_ports_list_length
```

Number of elements in the **NDDS_Transport_UDPv4_WAN_Property_t::comm_ports_list** (p. 2074).

[default] 0

9.343.2.21 port_offset

```
RTI_INT32 NDDS_Transport_UDPv4_WAN_Property_t::port_offset
```

Port offset to allow coexistence with built-in UDPv4 transport.

This property allows using the built-in UDPv4 transport and the Real-Time WAN Transport at the same time.

```
<transport_builtin>

    <mask>UDPv4_WAN|UDPv4</mask>

</transport_builtin>
```

When the UDP ports used by Real-Time WAN Transport are not explicitly set, they are calculated as follows: RTPS port + port_offset.

[default] 125

9.343.2.22 binding_ping_period

```
RTI_UINT32 NDDS_Transport_UDPv4_WAN_Property_t::binding_ping_period
```

Specifies the period in milliseconds at which BINDING PINGS messages are sent to keep NAT mappings open.

Configures the period in milliseconds at which BINDING_PING messages are sent by a local transport instance to a remote transport instance. For example, 1000 means to send BINDING_PING messages every second.

BINDING_PING messages are used on the sending side to open NAT bindings from a local transport instance to a remote transport instance and they are sent periodically to keep the bindings open.

On the receiving side, BINDING_PINGS are used to calculate the public IP transport address of an UUID locator. This address will be used to send data to the locator.

For additional information on the role of BINDING_PING, see the Core Libraries User's Manual.

From a configuration point of view, and to avoid communication disruptions, the period at which a transport instance sends BINDING_PING messages should be smaller than the NAT binding session timeout. This timeout depends on the NAT router configuration.

[default] 1000 (1 sec)

9.344 NDDS_Transport_UDPv6_Property_t Struct Reference

Configurable IPv6/UDP Transport-Plugin properties.

```
#include <transport_udpv6.h>
```

Public Attributes

- struct **NDDS_Transport_Property_t parent**

  Generic properties of all transport plugins.
- RTI_INT32 **send_socket_buffer_size**

  Size in bytes of the send buffer of a socket used for sending.
- RTI_INT32 **recv_socket_buffer_size**

  Size in bytes of the receive buffer of a socket used for receiving.
- RTI_INT32 **unicast_enabled**

  Allows the transport plugin to use unicast for sending and receiving.
- RTI_INT32 **multicast_enabled**

  Allows the transport plugin to use multicast for sending and receiving.
- RTI_INT32 **multicast_ttl**

  Value for the time-to-live parameter for all multicast sends using this plugin.
- RTI_INT32 **multicast_loopback_disabled**

  Prevents the transport plugin from putting multicast packets onto the loopback interface.
- RTI_INT32 **ignore_loopback_interface**

  Prevents the transport plugin from using the IP loopback interface.
- RTI_INT32 **ignore_nonrunning_interfaces**

  Prevents the transport plugin from using a network interface that is not reported as RUNNING by the operating system.
- RTI_INT32 **no_zero_copy**

  ***[DEPRECATED]** Prevents the transport plugin from doing zero copy.*
- RTI_INT32 **send_blocking**

  Control blocking behavior of send sockets. CHANGING THIS FROM THE DEFAULT CAN CAUSE SIGNIFICANT PERFORMANCE PROBLEMS.
- RTI_INT32 **enable_v4mapped**

  Specify whether UDPv6 transport will process IPv4 addresses.
- RTI_UINT32 **transport_priority_mask**

  Set mask for use of transport priority field.
- RTI_INT32 **transport_priority_mapping_low**

  Set low value of output range to IPv6 TCLASS.
- RTI_INT32 **transport_priority_mapping_high**

  Set high value of output range to IPv6 TCLASS.
- RTI_INT32 **send_ping**

  Configures whether to send PING messages.
- RTI_INT32 **force_interface_poll_detection**

  Forces the interface tracker to use a polling mechanism to detect changes on the UDPv6 interfaces.
- RTI_UINT32 **interface_poll_period**

  Specifies the period in milliseconds to query for changes in the state of all the interfaces.
- RTI_INT32 **reuse_multicast_receive_resource**

Controls whether or not to reuse multicast receive resources.

- RTI_INT32 **protocol_overhead_max**

  Maximum size in bytes of protocol overhead, including headers.

- RTI_INT32 **disable_interface_tracking**

  Disables detection of network interface changes.

- RTI_UINT32 **join_multicast_group_timeout**

  [Windows only] Defines how much time (milliseconds) to wait to join a multicast group address when a new interface is detected.

- char * **public_address**

  Public IP address associated with the transport instantiation.

- RTI_INT32 **port_offset**

  *Port offset to allow multiple instances of UDPv6 transports in the same **DDSDomainParticipant** (p. 1619).*

9.344.1 Detailed Description

Configurable IPv6/UDP Transport-Plugin properties.

You can modify the properties in this structure to configure the plugin. However, you must set the properties before the plugin is instantiated.

See also

NDDSTransportSupport::set_builtin_transport_property() (p. 2109)

NDDS_Transport_UDPv6_new (p. 486)

9.344.2 Member Data Documentation

9.344.2.1 parent

`struct` **NDDS_Transport_Property_t** NDDS_Transport_UDPv6_Property_t::parent

Generic properties of all transport plugins.

9.344.2.2 send_socket_buffer_size

```
RTI_INT32 NDDS_Transport_UDPv6_Property_t::send_socket_buffer_size
```

Size in bytes of the send buffer of a socket used for sending.

On most operating systems, `setsockopt()` will be called to set the SENDBUF to the value of this parameter.

This value must be greater than or equal to **NDDS_Transport_Property_t::message_size_max** (p. 2047). Its maximum value is dependent on the operating system.

By default, it will be set to **NDDS_TRANSPORT_UDPV6_SEND_SOCKET_BUFFER_SIZE_DEFAULT** (p. 484).

The maximum value for this property depends on the operating system.

Valid values are:

- **NDDS_TRANSPORT_UDPV6_SEND_SOCKET_BUFFER_SIZE_DEFAULT** (p. 484): `setsockopt()` (or equivalent) will be called to size the send buffer of the socket with the default value.
- **NDDS_TRANSPORT_UDPV6_SOCKET_BUFFER_SIZE_OS_DEFAULT** (p. 484): `setsockopt()` (or equivalent) will not be called to size the send buffer of the socket. RTI Connext will use the OS default.
- **NDDS_TRANSPORT_UDPV6_SOCKET_BUFFER_SIZE_OS_MAX** (p. 483): RTI Connext will retrieve the maximum socket buffer size allowed by the operating system and use that value for the send buffer size of the socket.
- Positive integer: `setsockopt()` (or equivalent) will be called to size the send buffer of the socket with the provided value. If this value exceeds the maximum allowed by the operating system, the operating system's maximum value will be used instead.

9.344.2.3 recv_socket_buffer_size

```
RTI_INT32 NDDS_Transport_UDPv6_Property_t::recv_socket_buffer_size
```

Size in bytes of the receive buffer of a socket used for receiving.

On most operating systems, `setsockopt()` will be called to set the RECVBUF to the value of this parameter.

This value must be greater than or equal to **NDDS_Transport_Property_t::message_size_max** (p. 2047). Its maximum value is dependent on the operating system.

By default, it will be set to **NDDS_TRANSPORT_UDPV6_RECV_SOCKET_BUFFER_SIZE_DEFAULT** (p. 484).

The maximum value for this property depends on the operating system.

Valid values are:

- **NDDS_TRANSPORT_UDPV6_RECV_SOCKET_BUFFER_SIZE_DEFAULT** (p. 484): `setsockopt()` (or equivalent) will be called to size the receive buffer of the socket with the default value.
- **NDDS_TRANSPORT_UDPV6_SOCKET_BUFFER_SIZE_OS_DEFAULT** (p. 484): `setsockopt()` (or equivalent) will not be called to size the receive buffer of the socket. RTI Connext will use the OS default.
- **NDDS_TRANSPORT_UDPV6_SOCKET_BUFFER_SIZE_OS_MAX** (p. 483): RTI Connext will retrieve the maximum socket buffer size allowed by the operating system and use that value for the receive buffer size of the socket.
- Positive integer: `setsockopt()` (or equivalent) will be called to size the receive buffer of the socket with the provided value. If this value exceeds the maximum allowed by the operating system, the operating system's maximum value will be used instead.

9.344.2.4 unicast_enabled

```
RTI_INT32 NDDS_Transport_UDPv6_Property_t::unicast_enabled
```

Allows the transport plugin to use unicast for sending and receiving.

This value turns unicast UDP on (if set to 1) or off (if set to 0) for this plugin. By default, it will be turned on (1). Also by default, the plugin will use all the allowed network interfaces that it finds up and running when the plugin is instantiated.

9.344.2.5 multicast_enabled

```
RTI_INT32 NDDS_Transport_UDPv6_Property_t::multicast_enabled
```

Allows the transport plugin to use multicast for sending and receiving.

This value turns multicast UDP on (if set to 1) or off (if set to 0) for this plugin. By default, it will be turned on (1) for those platforms that support multicast. Also by default, the plugin will use the all network interfaces allowed for multicast that it finds up and running when the plugin is instantiated.

9.344.2.6 multicast_ttl

```
RTI_INT32 NDDS_Transport_UDPv6_Property_t::multicast_ttl
```

Value for the time-to-live parameter for all multicast sends using this plugin.

This is used to set the TTL of multicast packets sent by this transport plugin.

[default] 1

See also

NDDS_TRANSPORT_UDPV6_MULTICAST_TTL_DEFAULT (p. 485)

9.344.2.7 multicast_loopback_disabled

```
RTI_INT32 NDDS_Transport_UDPv6_Property_t::multicast_loopback_disabled
```

Prevents the transport plugin from putting multicast packets onto the loopback interface.

If multicast loopback is disabled (this value is set to 1), then when sending multicast packets, RTI Connext will *not* put a copy of the packets on the loopback interface. This prevents applications on the same node (including itself) from receiving those packets.

This value is set to 0 by default, meaning multicast loopback is *enabled*.

Disabling multicast loopback (setting this value to 1) may result in minor performance gains when using multicast.

9.344.2.8 ignore_loopback_interface

```
RTI_INT32 NDDS_Transport_UDPv6_Property_t::ignore_loopback_interface
```

Prevents the transport plugin from using the IP loopback interface.

Currently three values are allowed:

- **0**: Forces local traffic to be sent over loopback, even if a more efficient transport (such as shared memory) is installed (in which case traffic will be sent over both transports).
- **1**: Disables local traffic via this plugin. Do not use the IP loopback interface even if no NICs are discovered. This is useful when you want applications running on the same node to use a more efficient transport (such as shared memory) instead of the IP loopback.
- **-1**: Automatic. Lets RTI Connext decide between the above two choices.

The current "automatic" (-1) RTI Connext policy is as follows.

- If a shared memory transport plugin is available for local traffic and there is a locator on the initial peers list that can use shared memory, the effective value is 1 (i.e., disable UDPv6 local traffic).
- Otherwise, the effective value is 0 (i.e., use UDPv6 for local traffic also).

[default] -1 Automatic RTI Connext policy based on availability of the shared memory transport.

9.344.2.9 ignore_nonrunning_interfaces

```
RTI_INT32 NDDS_Transport_UDPv6_Property_t::ignore_nonrunning_interfaces
```

Prevents the transport plugin from using a network interface that is not reported as RUNNING by the operating system.

The transport checks the flags reported by the operating system for each network interface upon initialization. An interface that is not reported as UP will not be used. This property allows the same check to be extended to the IFF_↩ RUNNING flag implemented by some operating systems. The RUNNING flag is defined to mean that "all resources are allocated," and may be off if there is no link detected, e.g., the network cable is unplugged.

Two values are allowed:

- **0**: Do not check the RUNNING flag when enumerating interfaces, just make sure interface is UP.
- **1**: Check flag when enumerating interfaces and ignore those that are not reported as RUNNING. This can be used on some operating systems to cause the transport to ignore interfaces that are enabled but not connected to the network.

[default] 1 (i.e., check RUNNING flag)

9.344.2.10 no_zero_copy

```
RTI_INT32 NDDS_Transport_UDPv6_Property_t::no_zero_copy
```

[DEPRECATED] Prevents the transport plugin from doing zero copy.

DEPRECATED: This property has no effect. By default, this plugin will use the zero copy on OSes that offer it. While this is good for performance, it may sometimes tax the OS resources in a manner that cannot be overcome by the application.

The best example is if the hardware/device driver lends the buffer to the application itself. If the application does not return the loaned buffers soon enough, the node may error or malfunction. If you cannot reconfigure the H/W, device driver, or the OS to allow the zero copy feature to work for your application, you may have no choice but to turn off the use of zero copy.

By default this is set to 0, so RTI Connext will use the zero copy API if offered by the OS.

9.344.2.11 send_blocking

```
RTI_INT32 NDDS_Transport_UDPv6_Property_t::send_blocking
```

Control blocking behavior of send sockets. CHANGING THIS FROM THE DEFAULT CAN CAUSE SIGNIFICANT PERFORMANCE PROBLEMS.

Currently two values are defined:

- **NDDS_TRANSPORT_UDPV6_BLOCKING_ALWAYS:** Sockets are blocking (default socket options for the operating system).
- **NDDS_TRANSPORT_UDPV6_BLOCKING_NEVER:** Sockets are modified to make them non-blocking. THIS MAY CAUSE SIGNIFICANT PERFORMANCE PROBLEMS.

[default] NDDS_TRANSPORT_UDPV6_BLOCKING_ALWAYS.

9.344.2.12 enable_v4mapped

```
RTI_INT32 NDDS_Transport_UDPv6_Property_t::enable_v4mapped
```

Specify whether UDPv6 transport will process IPv4 addresses.

Set this to 1 to turn on processing of IPv4 addresses. Note that this may make it incompatible with use of the UDPv4 transport within the same domain participant.

[default] 0.

9.344.2.13 transport_priority_mask

```
RTI_UINT32 NDDS_Transport_UDPv6_Property_t::transport_priority_mask
```

Set mask for use of transport priority field.

If transport priority mapping is supported on the platform, this mask is used in conjunction with **NDDS_↩ Transport_UDPv6_Property_t::transport_priority_mapping_low** (p. 2083) and **NDDS_Transport_UDPv6_↩ Property_t::transport_priority_mapping_high** (p. 2083) to define the mapping from the DDS transport priority (see **TRANSPORT_PRIORITY** (p. 647)) to the IPv6 TCLASS field. Defines a contiguous region of bits in the 32-bit transport priority value that is used to generate values for the IPv6 TCLASS field on an outgoing socket.

For example, the value 0x0000ff00 causes bits 9-16 (8 bits) to be used in the mapping. The value will be scaled from the mask range (0x0000 - 0xff00 in this case) to the range specified by low and high.

If the mask is set to zero, then the transport will not set IPv6 TCLASS for send sockets.

[default] 0.

9.344.2.14 transport_priority_mapping_low

```
RTI_INT32 NDDS_Transport_UDPv6_Property_t::transport_priority_mapping_low
```

Set low value of output range to IPv6 TCLASS.

This is used in conjunction with **NDDS_Transport_UDPv6_Property_t::transport_priority_mask** (p. 2082) and **NDDS_Transport_UDPv6_Property_t::transport_priority_mapping_high** (p. 2083) to define the mapping from the DDS transport priority to the IPv6 TCLASS field. Defines the low value of the output range for scaling.

Note that IPv6 TCLASS is generally an 8-bit value.

[default] 0.

9.344.2.15 transport_priority_mapping_high

```
RTI_INT32 NDDS_Transport_UDPv6_Property_t::transport_priority_mapping_high
```

Set high value of output range to IPv6 TCLASS.

This is used in conjunction with **NDDS_Transport_UDPv6_Property_t::transport_priority_mask** (p. 2082) and **NDDS_Transport_UDPv6_Property_t::transport_priority_mapping_low** (p. 2083) to define the mapping from the DDS transport priority to the IPv6 TCLASS field. Defines the high value of the output range for scaling.

Note that IPv6 TCLASS is generally an 8-bit value.

[default] 0xff.

9.344.2.16 send_ping

```
RTI_INT32 NDDS_Transport_UDPv6_Property_t::send_ping
```

Configures whether to send PING messages.

See also

NDDS_Transport_UDPv4_Property_t::send_ping (p. 2063)

9.344.2.17 force_interface_poll_detection

```
RTI_INT32 NDDS_Transport_UDPv6_Property_t::force_interface_poll_detection
```

Forces the interface tracker to use a polling mechanism to detect changes on the UDPv6 interfaces.

See also

NDDS_Transport_UDPv4_Property_t::force_interface_poll_detection (p. 2064)

9.344.2.18 interface_poll_period

```
RTI_UINT32 NDDS_Transport_UDPv6_Property_t::interface_poll_period
```

Specifies the period in milliseconds to query for changes in the state of all the interfaces.

See also

NDDS_Transport_UDPv4_Property_t::interface_poll_period (p. 2064)

9.344.2.19 reuse_multicast_receive_resource

```
RTI_INT32 NDDS_Transport_UDPv6_Property_t::reuse_multicast_receive_resource
```

Controls whether or not to reuse multicast receive resources.

See also

NDDS_Transport_UDPv4_Property_t::reuse_multicast_receive_resource (p. 2064)

9.344.2.20 protocol_overhead_max

```
RTI_INT32 NDDS_Transport_UDPv6_Property_t::protocol_overhead_max
```

Maximum size in bytes of protocol overhead, including headers.

This value is the maximum size, in bytes, of protocol-related overhead. Normally, the overhead accounts for UDP and IP headers. The default value is set to accommodate the most common UDP/IP header size.

Note that when **NDDS_Transport_Property_t::message_size_max** (p. 2047) plus this overhead is larger than the UDPv6 maximum message size (65535 bytes), the middleware will automatically reduce the effective message_size↩ _max, to 65535 minus this overhead.

[default] 48.

See also

NDDS_Transport_Property_t::message_size_max (p. 2047)

9.344.2.21 disable_interface_tracking

```
RTI_INT32 NDDS_Transport_UDPv6_Property_t::disable_interface_tracking
```

Disables detection of network interface changes.

See also

NDDS_Transport_UDPv4_Property_t::disable_interface_tracking (p. 2065)

9.344.2.22 join_multicast_group_timeout

```
RTI_UINT32 NDDS_Transport_UDPv6_Property_t::join_multicast_group_timeout
```

[Windows only] Defines how much time (milliseconds) to wait to join a multicast group address when a new interface is detected.

See also

NDDS_Transport_UDPv4_Property_t::join_multicast_group_timeout (p. 2065)

9.344.2.23 public_address

```
char* NDDS_Transport_UDPv6_Property_t::public_address
```

Public IP address associated with the transport instantiation.

Setting the public IP address is only necessary to support communication over WAN that involves Network Address Translation (NAT).

Typically, the address is the public address of the IP NAT router that provides access to the WAN.

By default, the **DDSDomainParticipant** (p. 1619) creating the transport will announce the IP addresses obtained from the NICs to other DomainParticipants in the system.

When this property is set, the DomainParticipant will announce the IP address corresponding to the property value instead of the LAN IP addresses associated with the NICs.

Note 1: Setting this property is necessary, but is not a sufficient condition for sending and receiving data over the WAN. You must also configure the IP NAT router to allow UDP traffic and to map the public IP address specified by this property to the DomainParticipant's private LAN IP address. This is typically done with one of the following mechanisms:

- Port Forwarding: You must map the private ports used to receive discovery and user data traffic to the corresponding public ports (see **DDS_RtpsWellKnownPorts_t** (p. 1338)). Public and private ports must be the same since the transport does not allow you to change the mapping.
- 1:1 NAT: You must add a 1:1 NAT entry that maps the public IP address specified in this property to the private LAN IP address of the DomainParticipant.

Note 2: By setting this property, the **DDSDomainParticipant** (p. 1619) only announces its public IP address to other DomainParticipants. Therefore, communication with DomainParticipants within the LAN that are running on different nodes will not work unless the NAT router is configured to enable NAT reflection (hairpin NAT).

There is another way to achieve simultaneous communication with DomainParticipants running in the LAN and WAN, that does not require hairpin NAT. This way uses a gateway application such as RTI Routing Service to provide access to the WAN.

[default] NULL (the transport uses the IP addresses obtained from the NICs)

9.344.2.24 port_offset

```
RTI_INT32 NDDS_Transport_UDPv6_Property_t::port_offset
```

Port offset to allow multiple instances of UDPv6 transports in the same **DDSDomainParticipant** (p. 1619).

By default, it is not possible to have multiple instances of the UDPv6 transport in the same **DDSDomainParticipant** (p. 1619). This is because all instances will try to bind to the same UDP port(s) for receiving data.

With this property, you can specify an offset that will be added to the RTPS port(s) to determine the UDP port(s) to bind to. This way, you can have multiple instances of the UDPv6 transport in the same **DDSDomainParticipant** (p. 1619).

An RTPS port will be mapped to an UDP port as follows: UDP port = RTPS port + port_offset.

[default] 0

9.345 NDDS_Transport_UUID Struct Reference

Unequivocally identifies a transport plugin instance.

`#include <transport_common.h>`

9.345.1 Detailed Description

Unequivocally identifies a transport plugin instance.

9.346 NDDS_Utility_NetworkCaptureParams_t Struct Reference

Input parameters for starting network capture.

Public Attributes

- struct **DDS_StringSeq transports**
  List of transports to capture.
- **NDDS_Utility_NetworkCaptureContentKindMask dropped_content**
  Exclude contents from the capture file.
- **NDDS_Utility_NetworkCaptureTrafficKindMask traffic**
  Traffic direction to capture.
- **DDS_Boolean parse_encrypted_content**
  If secure traffic should be decrypted or not.
- struct **DDS_ThreadSettings_t checkpoint_thread_settings**
  The properties of the checkpoint thread.
- **DDS_Long frame_queue_size**
  Size of the frame queue (Bytes).

9.346.1 Detailed Description

Input parameters for starting network capture.

9.346.2 Member Data Documentation

9.346.2.1 transports

struct **DDS_StringSeq** NDDS_Utility_NetworkCaptureParams_t::transports

List of transports to capture.

Network Capture will only save RTPS frames if the associated transport protocol is part of this sequence.

[default] Empty sequence initializer. This means that by default all transports will be captured.

9.346.2.2 dropped_content

NDDS_Utility_NetworkCaptureContentKindMask NDDS_Utility_NetworkCaptureParams_t::dropped_content

Exclude contents from the capture file.

It accepts values from **NDDS_Utility_NetworkCaptureContentKind** (p. 752) and **NDDS_Utility_NetworkCapture↩ ContentKindMask** (p. 751) .

We can choose to exclude user data or encrypted content from the capture file.

[default] No content is excluded - **NDDS_UTILITY_NETWORK_CAPTURE_CONTENT_MASK_DEFAULT** (p. 750).

9.346.2.3 traffic

NDDS_Utility_NetworkCaptureTrafficKindMask NDDS_Utility_NetworkCaptureParams_t::traffic

Traffic direction to capture.

It accepts values from **NDDS_Utility_NetworkCaptureTrafficKind** (p. 752) and **NDDS_Utility_NetworkCapture↩ TrafficKindMask** (p. 751) .

[default] Capture both inbound and outbound traffic - **NDDS_UTILITY_NETWORK_CAPTURE_TRAFFIC_MASK_↩ DEFAULT** (p. 750).

9.346.2.4 parse_encrypted_content

DDS_Boolean NDDS_Utility_NetworkCaptureParams_t::parse_encrypted_content

If secure traffic should be decrypted or not.

Network Capture supports decryption of RTPS messages and submessages. It does not support decryption of the user data payload (<data_protection_kind> tag in the GovernanceDocument).

[default] DDS_BOOLEAN_FALSE (p. 513)

9.346.2.5 checkpoint_thread_settings

```
struct DDS_ThreadSettings_t NDDS_Utility_NetworkCaptureParams_t::checkpoint_thread_settings
```

The properties of the checkpoint thread.

The checkpoint thread is a per-participant thread responsible for reading frames from a queue and saving them to disk (as soon as they are produced).

The members that can be configured are:

- **DDS_ThreadSettings_t::mask** (p. 1386). Default value of (**DDS_THREAD_SETTINGS_PRIORITY_ENFORCE** (p. 548) | **DDS_THREAD_SETTINGS_STDIO** (p. 548)).
- **DDS_ThreadSettings_t::priority** (p. 1386). Platform-dependent - Consult `Platform Notes` for additional details.
- **DDS_ThreadSettings_t::stack_size** (p. 1386). Platform-dependent - Consult `Platform Notes` for additional details.

9.346.2.6 frame_queue_size

```
DDS_Long NDDS_Utility_NetworkCaptureParams_t::frame_queue_size
```

Size of the frame queue (Bytes).

Network Capture enqueues frames before saving them to disk, which takes place in a separate thread.

Network Capture does not block if the queue becomes full. Attempting to enqueue a frame with a full frame queue will fail (frame won't be captured) with a log message.

The size of the queue is dependent on the network traffic (amount of frames that we want to capture) and system resources (how fast we can capture frames).

[default] 2097152 (2MB).

9.347 NDDSConfigActivityContext Class Reference

Activity Context APIs.

Static Public Member Functions

- static void **set_attribute_mask** (**NDDS_Config_ActivityContextAttributeKindMask** attribute_format)

  *Set the **NDDS_Config_ActivityContextAttributeKindMask** (p. 731) of the Activity Context.*

9.347.1 Detailed Description

Activity Context APIs.

Activity Context APIs.

9.348 NDDSConfigCompliance Class Reference

APIs to configure all the compliance mask.

Static Public Member Functions

- static **NDDS_Config_XTypesComplianceMask get_xtypes_mask** ()
  Gets the XTypes compliance mask value for the application.
- static void **set_xtypes_mask** (**NDDS_Config_XTypesComplianceMask** mask)
  Sets the level of compliance of the application with the XTypes specification.
- static void **load_compliance_masks** ()
  Load the compliance masks from the environment variables.
- static **NDDS_Config_RpcComplianceMask get_rpc_mask** ()
  Gets the RPC compliance mask value for the application.
- static void **set_rpc_mask** (**NDDS_Config_RpcComplianceMask** mask)
  Sets the level of compliance of the application with the DDS RPC specification.

9.348.1 Detailed Description

APIs to configure all the compliance mask.

9.348.2 Member Function Documentation

9.348.2.1 get_xtypes_mask()

```
static NDDS_Config_XTypesComplianceMask NDDSConfigCompliance::get_xtypes_mask ( ) [static]
```

Gets the XTypes compliance mask value for the application.

The **NDDS_Config_XTypesComplianceMask** (p. 736) configures different aspects of the Extended CDR encoding (XCDR and XCDR2) for the application.

Returns

The **NDDS_Config_XTypesComplianceMask** (p. 736) value.

See also

NDDS_Config_XTypesComplianceMask (p. 736)

NDDSConfigCompliance::get_xtypes_mask (p. 2090)

NDDSConfigCompliance::load_compliance_masks (p. 2091)

9.348.2.2 set_xtypes_mask()

```
static void NDDSConfigCompliance::set_xtypes_mask (
              NDDS_Config_XTypesComplianceMask mask )  [static]
```

Sets the level of compliance of the application with the XTypes specification.

The **NDDS_Config_XTypesComplianceMask** (p. 736) configures different aspects of the Extended CDR encoding (XCDR and XCDR2) for the application.

Once you call this function the value of the mask will apply to all the entities created after the call. It won't apply to entities created before the call.

In general, applications should only set the mask once, before executing any other RTI Connext API. The mask can also be set using the environment variable **NDDS_XTYPES_COMPLIANCE_MASK**.

Parameters

mask	<<***in***>> *(p. 433)* Compliance mask.

See also

NDDS_Config_XTypesComplianceMask (p. 736)

NDDSConfigCompliance::get_xtypes_mask (p. 2090)

NDDSConfigCompliance::load_compliance_masks (p. 2091)

9.348.2.3 load_compliance_masks()

```
static void NDDSConfigCompliance::load_compliance_masks ( )  [static]
```

Load the compliance masks from the environment variables.

This function loads the XTypes and RPC compliance masks from the environment, which can be set using the environment variables **NDDS_XTYPES_COMPLIANCE_MASK** and **NDDS_RPC_COMPLIANCE_MASK**.

See also

NDDS_Config_XTypesComplianceMask (p. 736)

NDDSConfigCompliance::set_xtypes_mask (p. 2090)

NDDSConfigCompliance::get_xtypes_mask (p. 2090)

NDDS_Config_RpcComplianceMask (p. 737)

NDDSConfigCompliance::set_rpc_mask (p. 2092)

NDDSConfigCompliance::get_rpc_mask (p. 2091)

9.348.2.4 get_rpc_mask()

```
static NDDS_Config_RpcComplianceMask NDDSConfigCompliance::get_rpc_mask ( ) [static]
```

Gets the RPC compliance mask value for the application.

The **NDDS_Config_RpcComplianceMask** (p. 737) configures different aspects of RPC communication.

Returns

The **NDDS_Config_RpcComplianceMask** (p. 737) value.

See also

NDDS_Config_RpcComplianceMask (p. 737)

NDDSConfigCompliance::get_rpc_mask (p. 2091)

NDDSConfigCompliance::load_compliance_masks (p. 2091)

9.348.2.5 set_rpc_mask()

```
static void NDDSConfigCompliance::set_rpc_mask (
            NDDS_Config_RpcComplianceMask mask ) [static]
```

Sets the level of compliance of the application with the DDS RPC specification.

The **NDDS_Config_RpcComplianceMask** (p. 737) configures different aspects of RPC communication.

Once you call this function the value of the mask will apply to all the entities created after the call. It won't apply to entities created before the call.

In general, applications should only set the mask once, before executing any other RTI Connext API. The mask can also be set using the environment variable **NDDS_RPC_COMPLIANCE_MASK**.

Parameters

mask	<<***in***>> *(p. 433)* Compliance mask.

See also

NDDS_Config_RpcComplianceMask (p. 737)

NDDSConfigCompliance::get_rpc_mask (p. 2091)

NDDSConfigCompliance::load_compliance_masks (p. 2091)

9.349 NDDSConfigLogger Class Reference

<<***interface***>> *(*p. *433)* The singleton type used to configure RTI Connext logging.

Public Member Functions

- **NDDS_Config_LogVerbosity get_verbosity** ()

  Get the verbosity at which RTI Connext is currently logging diagnostic information.
- **NDDS_Config_LogVerbosity get_verbosity_by_category** (**NDDS_Config_LogCategory** category)

  Get the verbosity at which RTI Connext is currently logging diagnostic information in the given category.
- void **set_verbosity** (**NDDS_Config_LogVerbosity** verbosity)

  Set the verbosity at which RTI Connext will log diagnostic information.
- void **set_verbosity_by_category** (**NDDS_Config_LogCategory** category, **NDDS_Config_LogVerbosity** verbosity)

  Set the verbosity at which RTI Connext will log diagnostic information in the given category.
- FILE * **get_output_file** ()

  Get the file to which the logged output is redirected.
- bool **set_output_file** (FILE *out)

  Set the file to which the logged output is redirected.
- bool **set_output_file_set** (const char *file_prefix, const char *file_suffix, int max_capacity, int max_files)

  Configure a set of files to redirect the logged output.
- **NDDSConfigLoggerDevice** * **get_output_device** ()

  Return the user device registered with the logger.
- bool **set_output_device** (**NDDSConfigLoggerDevice** *device)

  *Register a **NDDSConfigLoggerDevice** (*p. *2100).*
- **NDDS_Config_LogPrintFormat get_print_format** ()

  *Get the current message format for the log level **NDDS_CONFIG_LOG_LEVEL_ERROR** (*p. *724).*
- bool **set_print_format** (**NDDS_Config_LogPrintFormat** print_format)

  *Set the message format that RTI Connext will use to log diagnostic information for all the log levels, except for **NDDS_CONFIG_LOG_LEVEL_FATAL_ERROR** (*p. *724). When the **Activity Context** (*p. *729) is printed, the user can select the information that will be part of the **Activity Context** (*p. *729) by using the API **NDDSConfigActivityContext::set_attribute_mask** (*p. *733).*
- **NDDS_Config_LogPrintFormat get_print_format_by_log_level** (**NDDS_Config_LogLevel** log_level)

  Get the current message format, by log level, that RTI Connext is using to log diagnostic information.
- bool **set_print_format_by_log_level** (**NDDS_Config_LogPrintFormat** print_format, **NDDS_Config_LogLevel** log_level)

  *Set the message format, by log level, that RTI Connext will use to log diagnostic information. When the **Activity Context** (*p. *729) is printed, the user can select the information that will be part of the **Activity Context** (*p. *729) by using the API **NDDSConfigActivityContext::set_attribute_mask** (*p. *733).*
- void **emergency** (const char *msg)

  *Logs message with **NDDS_CONFIG_SYSLOG_LEVEL_EMERGENCY** (*p. *725) and **NDDS_CONFIG_LOG_CATEGORY_USER** (*p. *726)..*
- void **alert** (const char *msg)

  *Logs message with **NDDS_CONFIG_SYSLOG_LEVEL_ALERT** (*p. *725) and **NDDS_CONFIG_LOG_CATEGORY_USER** (*p. *726)..*
- void **critical** (const char *msg)

  *Logs message with **NDDS_CONFIG_SYSLOG_LEVEL_CRITICAL** (*p. *725) and **NDDS_CONFIG_LOG_CATEGORY_USER** (*p. *726)..*

- void **error** (const char *msg)

  *Logs message with **NDDS_CONFIG_SYSLOG_LEVEL_ERROR** (p. 725) and **NDDS_CONFIG_LOG_CATEGORY_USER** (p. 726)..*
- void **warning** (const char *msg)

  *Logs message with **NDDS_CONFIG_SYSLOG_LEVEL_WARNING** (p. 725) and **NDDS_CONFIG_LOG_CATEGORY_USER** (p. 726)..*
- void **notice** (const char *msg)

  *Logs message with **NDDS_CONFIG_SYSLOG_LEVEL_NOTICE** (p. 725) and **NDDS_CONFIG_LOG_CATEGORY_USER** (p. 726)..*
- void **informational** (const char *msg)

  *Logs message with **NDDS_CONFIG_SYSLOG_LEVEL_INFORMATIONAL** (p. 725) and **NDDS_CONFIG_LOG_CATEGORY_USER** (p. 726)..*
- void **debug** (const char *msg)

  *Logs message with **NDDS_CONFIG_SYSLOG_LEVEL_DEBUG** (p. 725) and **NDDS_CONFIG_LOG_CATEGORY_USER** (p. 726)..*

Static Public Member Functions

- static **NDDSConfigLogger** * **get_instance** ()

  Get the singleton instance of this type.
- static void **finalize_instance** ()

  Finalize the singleton instance of this type.

9.349.1 Detailed Description

<<***interface***>> *(p. 433)* The singleton type used to configure RTI Connext logging.

9.349.2 Member Function Documentation

9.349.2.1 get_instance()

```
static NDDSConfigLogger * NDDSConfigLogger::get_instance ( ) [static]
```

Get the singleton instance of this type.

Examples

HelloWorld_publisher.cxx, and **HelloWorld_subscriber.cxx**.

9.349.2.2 finalize_instance()

```
static void NDDSConfigLogger::finalize_instance ( )  [static]
```

Finalize the singleton instance of this type.

MT Safety:

UNSAFE. It is not safe to call this method while another thread may be simultaneously calling another logger-related method, including this one.

9.349.2.3 get_verbosity()

```
NDDS_Config_LogVerbosity NDDSConfigLogger::get_verbosity ( )
```

Get the verbosity at which RTI Connext is currently logging diagnostic information.

The default verbosity if **NDDSConfigLogger::set_verbosity** (p. 2095) is never called is **NDDS_CONFIG_LOG_↩ VERBOSITY_ERROR** (p. 724).

If **NDDSConfigLogger::set_verbosity_by_category** (p. 2095) has been used to set different verbosities for different categories of messages, this method will return the maximum verbosity of all categories.

9.349.2.4 get_verbosity_by_category()

```
NDDS_Config_LogVerbosity NDDSConfigLogger::get_verbosity_by_category (
            NDDS_Config_LogCategory category )
```

Get the verbosity at which RTI Connext is currently logging diagnostic information in the given category.

The default verbosity if **NDDSConfigLogger::set_verbosity** (p. 2095) and **NDDSConfigLogger::set_verbosity_by↩ _category** (p. 2095) are never called is **NDDS_CONFIG_LOG_VERBOSITY_ERROR** (p. 724).

9.349.2.5 set_verbosity()

```
void NDDSConfigLogger::set_verbosity (
            NDDS_Config_LogVerbosity verbosity )
```

Set the verbosity at which RTI Connext will log diagnostic information.

Note: Logging at high verbosities will be detrimental to your application's performance. Your default setting should typically remain at **NDDS_CONFIG_LOG_VERBOSITY_WARNING** (p. 724) or below. (The default verbosity if you never set it is **NDDS_CONFIG_LOG_VERBOSITY_ERROR** (p. 724).)

Examples

HelloWorld_publisher.cxx, and **HelloWorld_subscriber.cxx**.

9.349.2.6 set_verbosity_by_category()

```
void NDDSConfigLogger::set_verbosity_by_category (
            NDDS_Config_LogCategory category,
            NDDS_Config_LogVerbosity verbosity )
```

Set the verbosity at which RTI Connext will log diagnostic information in the given category.

9.349.2.7 get_output_file()

```
FILE * NDDSConfigLogger::get_output_file ( )
```

Get the file to which the logged output is redirected.

If no output file has been registered through **NDDSConfigLogger::set_output_file** (p. 2096), this method will return NULL. In this case, logged output will on most platforms go to standard out as if through printf.

9.349.2.8 set_output_file()

```
bool NDDSConfigLogger::set_output_file (
            FILE * out )
```

Set the file to which the logged output is redirected.

The file passed may be NULL, in which case further logged output will be redirected to the platform-specific default output location (standard out on most platforms).

For better performance when log messages are generated frequently, the log messages are not flushed into a file immediately after they are generated. In other words, while writing a log message, RTI Connext only calls the function fwrite() (see https://pubs.opengroup.org/onlinepubs/009695399/functions/fwrite.html); it does not call the function fflush() (see https://pubs.opengroup.org/onlinepubs/009695399/functions/fflush.html). If your application requires a different flushing behavior, you may use **NDDSConfigLogger::set_output_device** (p. 2097) to configure a custom logging device.

9.349.2.9 set_output_file_set()

```
bool NDDSConfigLogger::set_output_file_set (
            const char * file_prefix,
            const char * file_suffix,
            int max_capacity,
            int max_files )
```

Configure a set of files to redirect the logged output.

The logged output will be redirected to a set of files whose names are configured with a prefix and a suffix. The maximum number of bytes configures how many bytes to write into a file before opening the next file. After reaching the maximum number of files, the first one is overwritten.

For example, if the prefix is '**Foo** (p. 1916)', the suffix is '.txt', the max number of bytes is 1GB, and the max number of files is 3, the logger will create (at most) these files: Foo1.txt, Foo2.txt, and Foo3.txt. It will write to Foo1.txt, and after writing 1GB, it will move on to Foo2.txt, then to Foo3.txt, then to Foo1.txt again, and so on.

To stop logging to these files and redirect the output to the platform-specific location, pass NULL, NULL, 0, 0.

See **NDDSConfigLogger::set_output_file** (p. 2096) for the flushing behavior.

9.349.2.10 get_output_device()

```
NDDSConfigLoggerDevice * NDDSConfigLogger::get_output_device ( )
```

Return the user device registered with the logger.

Returns

Registered user device or NULL if no user device is registered.

9.349.2.11 set_output_device()

```
bool NDDSConfigLogger::set_output_device (
            NDDSConfigLoggerDevice * device )
```

Register a **NDDSConfigLoggerDevice** (p. 2100).

Register the specified logging device with the logger.

There can be at most only one device registered with the logger at any given time.

When a device is installed, the logger will stop sending the log messages to the standard output and to the file set with **NDDSConfigLogger::set_output_file** (p. 2096).

To remove an existing device, use this method with NULL as the device parameter. After a device is removed the logger will continue sending log messages to the standard output and to the output file.

To replace an existing device with a new device, use this method providing the new device as the device parameter.

When a device is unregistered (by setting it to NULL), **NDDSConfigLoggerDevice** (p. 2100) calls the method **NDDSConfigLoggerDevice::close** (p. 2101).

Parameters

device	<<***in***>> *(p. 433)* Logging device.

9.349.2.12 get_print_format()

```
NDDS_Config_LogPrintFormat NDDSConfigLogger::get_print_format ( )
```

Get the current message format for the log level **NDDS_CONFIG_LOG_LEVEL_ERROR** (p. 724).

Use **NDDSConfigLogger::get_print_format_by_log_level** (p. 2098) to retrieve the format for other log levels.

If **NDDSConfigLogger::set_print_format** (p. 2097) is never called, the default format is **NDDS_CONFIG_LOG_↩ PRINT_FORMAT_DEFAULT** (p. 726).

9.349.2.13 set_print_format()

```
bool NDDSConfigLogger::set_print_format (
            NDDS_Config_LogPrintFormat print_format )
```

Set the message format that RTI Connext will use to log diagnostic information for all the log levels, except for **NDDS_CONFIG_LOG_LEVEL_FATAL_ERROR** (p. 724). When the **Activity Context** (p. 729) is printed, the user can select the information that will be part of the **Activity Context** (p. 729) by using the API **NDDSConfigActivityContext::set_attribute_mask** (p. 733).

9.349.2.14 get_print_format_by_log_level()

```
NDDS_Config_LogPrintFormat NDDSConfigLogger::get_print_format_by_log_level (
            NDDS_Config_LogLevel log_level )
```

Get the current message format, by log level, that RTI Connext is using to log diagnostic information.

If **NDDSConfigLogger::set_print_format** (p. 2097) is never called, the default format is **NDDS_CONFIG_LOG_PRINT_FORMAT_DEFAULT** (p. 726).

9.349.2.15 set_print_format_by_log_level()

```
bool NDDSConfigLogger::set_print_format_by_log_level (
            NDDS_Config_LogPrintFormat print_format,
            NDDS_Config_LogLevel log_level )
```

Set the message format, by log level, that RTI Connext will use to log diagnostic information. When the **Activity Context** (p. 729) is printed, the user can select the information that will be part of the **Activity Context** (p. 729) by using the API **NDDSConfigActivityContext::set_attribute_mask** (p. 733).

9.349.2.16 emergency()

```
void NDDSConfigLogger::emergency (
            const char * msg )
```

Logs message with **NDDS_CONFIG_SYSLOG_LEVEL_EMERGENCY** (p. 725) and **NDDS_CONFIG_LOG_CATEGORY_USER** (p. 726)..

9.349.2.17 alert()

```
void NDDSConfigLogger::alert (
            const char * msg )
```

Logs message with **NDDS_CONFIG_SYSLOG_LEVEL_ALERT** (p. 725) and **NDDS_CONFIG_LOG_CATEGORY_↩USER** (p. 726)..

9.349.2.18 critical()

```
void NDDSConfigLogger::critical (
            const char * msg )
```

Logs message with **NDDS_CONFIG_SYSLOG_LEVEL_CRITICAL** (p. 725) and **NDDS_CONFIG_LOG_CATEGORY↩_USER** (p. 726)..

9.349.2.19 error()

```
void NDDSConfigLogger::error (
            const char * msg )
```

Logs message with **NDDS_CONFIG_SYSLOG_LEVEL_ERROR** (p. 725) and **NDDS_CONFIG_LOG_CATEGORY_↩USER** (p. 726)..

9.349.2.20 warning()

```
void NDDSConfigLogger::warning (
            const char * msg )
```

Logs message with **NDDS_CONFIG_SYSLOG_LEVEL_WARNING** (p. 725) and **NDDS_CONFIG_LOG_↩CATEGORY_USER** (p. 726)..

9.349.2.21 notice()

```
void NDDSConfigLogger::notice (
            const char * msg )
```

Logs message with **NDDS_CONFIG_SYSLOG_LEVEL_NOTICE** (p. 725) and **NDDS_CONFIG_LOG_CATEGORY_↩USER** (p. 726)..

9.349.2.22 informational()

```
void NDDSConfigLogger::informational (
            const char * msg )
```

Logs message with **NDDS_CONFIG_SYSLOG_LEVEL_INFORMATIONAL** (p. 725) and **NDDS_CONFIG_LOG_↩ CATEGORY_USER** (p. 726)..

9.349.2.23 debug()

```
void NDDSConfigLogger::debug (
            const char * msg )
```

Logs message with **NDDS_CONFIG_SYSLOG_LEVEL_DEBUG** (p. 725) and **NDDS_CONFIG_LOG_CATEGORY_↩ USER** (p. 726)..

9.350 NDDSConfigLoggerDevice Class Reference

<<***interface***>> *(p. 433)* Logging device interface. Use for user-defined logging devices.

Public Member Functions

- virtual void **write** (const **NDDS_Config_LogMessage** *message)=0

  Write a log message to a specified logging device.
- virtual void **close** ()

  Close the logging device.

9.350.1 Detailed Description

<<***interface***>> *(p. 433)* Logging device interface. Use for user-defined logging devices.

Interface for handling log messages.

By default, the logger sends the log messages generated by RTI Connext to the standard output.

You can use the method **NDDSConfigLogger::set_output_file** (p. 2096) to redirect the log messages to a file.

To further customize the management of generated log messages, the logger offers the method **NDDSConfigLogger↩ ::set_output_device** (p. 2097) that allows you to install a user-defined logging device.

The logging device installed by the user must implement this interface.

Note: It is not safe to make any calls to the RTI Connext core library, including calls to **DDSDomainParticipant::get↩ _current_time** (p. 1666), from any of the logging device operations.

9.350.2 Member Function Documentation

9.350.2.1 write()

```
virtual void NDDSConfigLoggerDevice::write (
            const  NDDS_Config_LogMessage * message )  [pure virtual]
```

Write a log message to a specified logging device.

Note: It is not safe to make any calls to the RTI Connext core library, including calls to **DDSDomainParticipant::get↩ _current_time** (p. 1666), from any of the logging device operations.

Parameters

message	<<***in***>> *(p. 433)* Message to log.

9.350.2.2 close()

```
virtual void NDDSConfigLoggerDevice::close ( )  [virtual]
```

Close the logging device.

Note: It is not safe to make any calls to the RTI Connext core library, including calls to **DDSDomainParticipant::get↩ _current_time** (p. 1666), from any of the logging device operations.

9.351 NDDSConfigVersion Class Reference

<<***interface***>> *(p. 433)* The version of an RTI Connext distribution.

Public Member Functions

- const **DDS_ProductVersion_t** & **get_product_version** () const
  Get the RTI Connext product version.
- const **NDDS_Config_LibraryVersion_t** & **get_cpp_api_version** () const
  Get the version of the C++ API library.
- const **NDDS_Config_LibraryVersion_t** & **get_c_api_version** () const
  Get the version of the C API library.
- const **NDDS_Config_LibraryVersion_t** & **get_core_version** () const
  Get the version of the core library.
- const char * **to_string** () const
  Get this version in string form.

Static Public Member Functions

- static const **NDDSConfigVersion** & **get_instance** ()

  Get the singleton instance of this type.

9.351.1 Detailed Description

<<***interface***>> *(p. 433)* The version of an RTI Connext distribution.

The complete version is made up of the versions of the individual libraries that make up the product distribution.

9.351.2 Member Function Documentation

9.351.2.1 get_instance()

```
static const  NDDSConfigVersion & NDDSConfigVersion::get_instance ( )  [static]
```

Get the singleton instance of this type.

9.351.2.2 get_product_version()

```
const  DDS_ProductVersion_t & NDDSConfigVersion::get_product_version ( ) const
```

Get the RTI Connext product version.

9.351.2.3 get_cpp_api_version()

```
const  NDDS_Config_LibraryVersion_t & NDDSConfigVersion::get_cpp_api_version ( ) const
```

Get the version of the C++ API library.

9.351.2.4 get_c_api_version()

```
const  NDDS_Config_LibraryVersion_t & NDDSConfigVersion::get_c_api_version ( ) const
```

Get the version of the C API library.

9.351.2.5 get_core_version()

```
const  NDDS_Config_LibraryVersion_t & NDDSConfigVersion::get_core_version ( ) const
```

Get the version of the core library.

9.351.2.6 to_string()

```
const char * NDDSConfigVersion::to_string ( ) const
```

Get this version in string form.

Combine all of the constituent library versions into a single string.

The memory in which the string is stored in internal to this **NDDSConfigVersion** (p. 2101). The caller should not modify it.

9.352 NDDSLM Class Reference

Other Utilities APIs.

Static Public Member Functions

- static **DDS_Boolean unload_activation_library** ()

  Unloads the Activation Library (rti_activation) if it was previously loaded.

9.352.1 Detailed Description

Other Utilities APIs.

9.352.2 Member Function Documentation

9.352.2.1 unload_activation_library()

```
static  DDS_Boolean NDDSLM::unload_activation_library ( )  [static]
```

Unloads the Activation Library (rti_activation) if it was previously loaded.

This function should be called just before your process finishes. See the License Management section in the `Installation Guide` for additional details.

Returns

DDS_BOOLEAN_TRUE (p. 513) if successful. Otherwise, **DDS_BOOLEAN_FALSE** (p. 513)

9.353 NDDSTransportSupport Class Reference

<<***interface***>> *(*p. *433)* The utility class used to configure RTI Connext pluggable transports.

Static Public Member Functions

- static **NDDS_Transport_Handle_t** **register_transport** (**DDSDomainParticipant** *participant_in, NDDS_Transport_Plugin *transport_in, const **DDS_StringSeq** &aliases_in, const **NDDS_Transport_Address_t** &network_address_in)

  Register a transport plugin for use with a ***DDSDomainParticipant*** *(*p. *1619), assigning it a network_address.*
- static **NDDS_Transport_Handle_t** **lookup_transport** (**DDSDomainParticipant** *participant_in, **DDS_StringSeq** &aliases_out, **NDDS_Transport_Address_t** &network_address_out, NDDS_Transport_Plugin *transport_in)

  Look up a transport plugin within a ***DDSDomainParticipant*** *(*p. *1619).*
- static **DDS_ReturnCode_t** **add_send_route** (const **NDDS_Transport_Handle_t** &transport_handle_in, const **NDDS_Transport_Address_t** &address_range_in, **DDS_Long** address_range_bit_count_in)

  Add a route for outgoing messages.
- static **DDS_ReturnCode_t** **add_receive_route** (const **NDDS_Transport_Handle_t** &transport_handle_in, const **NDDS_Transport_Address_t** &address_range_in, **DDS_Long** address_range_bit_count_in)

  Add a route for incoming messages.
- static **DDS_ReturnCode_t** **get_builtin_transport_property** (**DDSDomainParticipant** *participant_in, **DDS_TransportBuiltinKind** builtin_transport_kind_in, struct **NDDS_Transport_Property_t** &builtin_transport_property_inout)

  Get the properties used to create a builtin transport plugin.
- static **DDS_ReturnCode_t** **set_builtin_transport_property** (**DDSDomainParticipant** *participant_in, **DDS_TransportBuiltinKind** builtin_transport_kind_in, const struct **NDDS_Transport_Property_t** &builtin_transport_property_in)

  Set the properties used to create a builtin transport plugin.
- static NDDS_Transport_Plugin * **get_transport_plugin** (**DDSDomainParticipant** *participant_in, const char *alias_in)

  Retrieve a transport plugin registered in a ***DDSDomainParticipant*** *(*p. *1619) by its alias.*

9.353.1 Detailed Description

<<***interface***>> *(*p. *433)* The utility class used to configure RTI Connext pluggable transports.

9.353.2 Member Function Documentation

9.353.2.1 register_transport()

```
static  NDDS_Transport_Handle_t NDDSTransportSupport::register_transport (
              DDSDomainParticipant * participant_in,
             NDDS_Transport_Plugin * transport_in,
             const  DDS_StringSeq & aliases_in,
             const  NDDS_Transport_Address_t & network_address_in )  [static]
```

Register a transport plugin for use with a **DDSDomainParticipant** (p. 1619), assigning it a network_address.

A transport plugin instance can be used by exactly one **DDSDomainParticipant** (p. 1619) at a time.

When a DataWriter/DataReader is created, only those transports already registered to the corresponding **DDSDomainParticipant** (p. 1619) are available to the DataWriter/DataReader.

Builtin transports can be automatically registered by RTI Connext as a convenience to the user. See **Built-in Transport Plugins** (p. 182) for details on how to control the builtin transports that are automatically registered.

Precondition

A disabled **DDSDomainParticipant** (p. 1619) and a transport plugin that will be registered exclusively with it.

Parameters

participant_in	<<***in***>> *(p. 433)* A non-null, disabled **DDSDomainParticipant** (p. 1619).
transport_in	<<***in***>> *(p. 433)* A non-null transport plugin that is currently not registered with another **DDSDomainParticipant** (p. 1619).
aliases_in	<<***in***>> *(p. 433)* A non-null sequence of strings used as aliases to symbolically refer to the transport plugins. The transport plugin will be "available for use" by a **DDSEntity** (p. 1731) in the **DDSDomainParticipant** (p. 1619) if the transport alias list associated with the **DDSEntity** (p. 1731) contains one of these transport aliases. An empty alias list represents a wildcard and matches all aliases. Alias names for the builtin transports are defined in **TRANSPORT_BUILTIN** (p. 641).
network_address_in	<<***in***>> *(p. 433)* The network address at which to register this transport plugin. The bits indicated by `transport_in.property.address_bit_count` will be truncated. The remaining bits are the network address of the transport plugin. (see **Transport Class Attributes** (p. 174)).

Returns

Upon success, a valid non-NIL transport handle, representing the association between the **DDSDomainParticipant** (p. 1619) and the transport plugin; a **NDDS_TRANSPORT_HANDLE_NIL** (p. 182) upon failure.

Note that a transport plugin's class name is automatically registered as an implicit alias for the plugin. Thus, a class name can be used to refer to all the transport plugin instance of that class.

See also

Transport Class Attributes (p. 174)

Transport Network Address (p. 176)

Locator Format (p. 663)

NDDS_DISCOVERY_PEERS (p. 662)

9.353.2.2 lookup_transport()

```
static NDDS_Transport_Handle_t NDDSTransportSupport::lookup_transport (
            DDSDomainParticipant * participant_in,
            DDS_StringSeq & aliases_out,
            NDDS_Transport_Address_t & network_address_out,
           NDDS_Transport_Plugin * transport_in ) [static]
```

Look up a transport plugin within a **DDSDomainParticipant** (p. 1619).

The transport plugin should have already been registered with the **DDSDomainParticipant** (p. 1619).

Parameters

participant_in	<<***in***>> *(*p. *433)* A non-null **DDSDomainParticipant** (p. 1619).
aliases_out	<<***inout***>> *(*p. *433)* A sequence of string where the aliases used to refer to the transport plugin symbolically will be returned. null if not interested.
network_address_out	<<***inout***>> *(*p. *433)* The network address at which to register the transport plugin will be returned here. null if not interested.
transport_in	<<***in***>> *(*p. *433)* A non-null transport plugin that is already registered with the **DDSDomainParticipant** (p. 1619).

Returns

Upon success, a valid non-NIL transport handle, representating the association between the **DDSDomain↩ Participant** (p. 1619) and the transport plugin; a **NDDS_TRANSPORT_HANDLE_NIL** (p. 182) upon failure.

See also

Transport Class Attributes (p. 174)

Transport Network Address (p. 176)

9.353.2.3 add_send_route()

```
static DDS_ReturnCode_t NDDSTransportSupport::add_send_route (
            const NDDS_Transport_Handle_t & transport_handle_in,
            const NDDS_Transport_Address_t & address_range_in,
             DDS_Long address_range_bit_count_in ) [static]
```

Add a route for outgoing messages.

This method can be used to narrow the range of addresses to which outgoing messages can be sent.

Precondition

A disabled **DDSDomainParticipant** (p. 1619).

Parameters

transport_handle_in	<<***in***>> (p. *433)* A valid non-NIL transport handle as a result of a call to **NDDSTransportSupport::register_transport()** (p. 2104).
address_range_in	<<***in***>> (p. *433)* The outgoing address range for which to use this transport plugin.
address_range_bit_count_in	<<***in***>> (p. *433)* The number of most significant bits used to specify the address range. This function only works for transport plugins that have a positive **NDDS_Transport_Property_t::address_bit_count** (p. 2046).

Returns

One of the **Standard Return Codes** (p. 532), or **DDS_RETCODE_PRECONDITION_NOT_MET** (p. 532).

See also

Transport Send Route (p. 176)

9.353.2.4 add_receive_route()

```
static DDS_ReturnCode_t NDDSTransportSupport::add_receive_route (
            const NDDS_Transport_Handle_t & transport_handle_in,
            const NDDS_Transport_Address_t & address_range_in,
             DDS_Long address_range_bit_count_in ) [static]
```

Add a route for incoming messages.

This method can be used to narrow the range of addresses at which to receive incoming messages.

Precondition

A disabled **DDSDomainParticipant** (p. 1619).

Parameters

transport_handle_in	<<***in***>> *(p. 433)* A valid non-NIL transport handle as a result of a call to **NDDSTransportSupport::register_transport()** (p. 2104).
address_range_in	<<***in***>> *(p. 433)* The incoming address range for which to use this transport plugin.
address_range_bit_count_in	<<***in***>> *(p. 433)* The number of most significant bits used to specify the address range. This function only works for transport plugins that have a positive **NDDS_Transport_Property_t::address_bit_count** (p. 2046).

Returns

One of the **Standard Return Codes** (p. 532), or **DDS_RETCODE_PRECONDITION_NOT_MET** (p. 532).

See also

Transport Receive Route (p. 177)

9.353.2.5 get_builtin_transport_property()

```
static DDS_ReturnCode_t NDDSTransportSupport::get_builtin_transport_property (
            DDSDomainParticipant * participant_in,
            DDS_TransportBuiltinKind builtin_transport_kind_in,
          struct NDDS_Transport_Property_t & builtin_transport_property_inout ) [static]
```

Get the properties used to create a builtin transport plugin.

Retrieves the properties that will be used to create a builtin transport plugin.

Precondition

The `builtin_transport_property_inout` parameter must be of the type specified by the `builtin_transport_kind_in`.

Parameters

participant_in	<<***in***>> *(p. 433)* A valid non-null **DDSDomainParticipant** (p. 1619)
builtin_transport_kind_in	<<***in***>> *(p. 433)* The builtin transport kind for which to retrieve the properties.
builtin_transport_property_inout	<<***inout***>> *(p. 433)* The storage area where the retrieved property will be output. The specific type required by the `builtin_transport_kind_in` must be used.

Exceptions

One	of the **Standard Return Codes** (p. 532), or **DDS_RETCODE_PRECONDITION_NOT_MET** (p. 532).

See also

NDDSTransportSupport::set_builtin_transport_property() (p. 2109)

9.353.2.6 set_builtin_transport_property()

```
static DDS_ReturnCode_t NDDSTransportSupport::set_builtin_transport_property (
            DDSDomainParticipant * participant_in,
            DDS_TransportBuiltinKind builtin_transport_kind_in,
            const struct NDDS_Transport_Property_t & builtin_transport_property_in ) [static]
```

Set the properties used to create a builtin transport plugin.

Specifies the properties that will be used to create a builtin transport plugin.

If the builtin transport is already registered when this operation is called, these property changes will *not* have any effect. Builtin transport properties should always be set before the transport is registered. See **Built-in Transport Plugins** (p. 182) for details on when a builtin transport is registered.

Precondition

A disabled **DDSDomainParticipant** (p. 1619). The `builtin_transport_property_inout` parameter must be of the type specified by the `builtin_transport_kind_in`.

Parameters

participant_in	<<***in***>> *(p. 433)* A valid non-null **DDSDomainParticipant** (p. 1619) that has not been enabled.
builtin_transport_kind_in	<<***in***>> *(p. 433)* The builtin transport kind for which to specificy the properties.
builtin_transport_property_in	<<***inout***>> *(p. 433)* The new transport property that will be used to the create the builtin transport plugin. The specific type required by the `builtin_transport_kind_in` must be used.

Exceptions

One	of the **Standard Return Codes** (p. 532), or **DDS_RETCODE_PRECONDITION_NOT_MET** (p. 532).

See also

NDDSTransportSupport::get_builtin_transport_property() (p. 2108)

9.353.2.7 get_transport_plugin()

```
static NDDS_Transport_Plugin * NDDSTransportSupport::get_transport_plugin (
            DDSDomainParticipant * participant_in,
            const char * alias_in )  [static]
```

Retrieve a transport plugin registered in a **DDSDomainParticipant** (p. 1619) by its alias.

This method can be used to get a pointer to a transport Plugin that has been registered into the **DDSDomainParticipant** (p. 1619).

Parameters

participant_in	<<***in***>> *(p. 433)* A non-null **DDSDomainParticipant** (p. 1619).
alias_in	<<***in***>> *(p. 433)* A non-null string used to symbolically refer to the transport plugins.

Returns

Upon success, a valid non-null pointer to a registered plugin; a null pointer if a plugin with that alias is not registered/found in that participant.

9.354 NDDSUtility Class Reference

Other Utilities APIs.

Static Public Member Functions

- static void **sleep** (const struct **DDS_Duration_t** &durationIn)
  Block the calling thread for the specified duration.
- static void **spin** (**DDS_UnsignedLongLong** spinCount)
  Performs the spin operation as many times as indicated.
- static **DDS_UnsignedLongLong** **get_spin_per_microsecond** ()
  Returns the number of spin operations to perform to wait 1 microsecond.
- static **DDS_Boolean** **enable_heap_monitoring** ()
  [DEPRECATED] *See:* ***NDDSUtilityHeapMonitoring::enable*** *(p. 2114).*
- static void **disable_heap_monitoring** ()
  [DEPRECATED] *See:* ***NDDSUtilityHeapMonitoring::disable*** *(p. 2114).*
- static **DDS_Boolean** **pause_heap_monitoring** ()
  [DEPRECATED] *See:* ***NDDSUtilityHeapMonitoring::pause*** *(p. 2115).*
- static **DDS_Boolean** **resume_heap_monitoring** ()
  [DEPRECATED] *See:* ***NDDSUtilityHeapMonitoring::resume*** *(p. 2115)*
- static **DDS_Boolean** **take_heap_snapshot** (const char *filename, **DDS_Boolean** print_details)
  [DEPRECATED] *See:* ***NDDSUtilityHeapMonitoring::take_heap_snapshot*** *(p. 2115).*

9.354.1 Detailed Description

Other Utilities APIs.

9.354.2 Member Function Documentation

9.354.2.1 sleep()

```
static void NDDSUtility::sleep (
            const struct  DDS_Duration_t & durationIn )  [static]
```

Block the calling thread for the specified duration.

Note that the achievable resolution of sleep is OS-dependent. That is, do not assume that you can sleep for 1 nanosecond just because you can specify a 1-nanosecond sleep duration via the API. The sleep resolution on most operating systems is usually 10 ms or greater.

Parameters

durationIn	<<***in***>> *(*p. *433)* Sleep duration.

MT Safety:

safe

Examples

HelloWorld_publisher.cxx.

9.354.2.2 spin()

```
static void NDDSUtility::spin (
             DDS_UnsignedLongLong spinCount )  [static]
```

Performs the spin operation as many times as indicated.

Spinning is the action of performing useless operations in a for loop in order to actively wait some time without yielding the CPU. Given that the resolution of sleep is in the order of ms, you can use this utility to wait times in the order of microseconds. To properly use this functionality, it is useful to measure previously the number of spin operations needed to wait the equivalent to microsecond (using the utility get_spin_per_microsecond) and then compute the corresponding spin count desired.

Parameters

spinCount	<<***in***>> *(p. 433)* Number of spin operations to perform.

9.354.2.3 get_spin_per_microsecond()

```
static DDS_UnsignedLongLong NDDSUtility::get_spin_per_microsecond ( ) [static]
```

Returns the number of spin operations to perform to wait 1 microsecond.

This utility can be used to measure how many spin operations must be performed to wait 1 microsecond. Since the time that it takes the CPU to perform 1 spin operation depends on the CPU frequency, it is recommended to use this utility before using **spin()** (p. 2111).

Returns

Number of spin operations to wait 1 microsecond.

See also

NDDSUtility::spin (p. 2111)

9.354.2.4 enable_heap_monitoring()

```
static DDS_Boolean NDDSUtility::enable_heap_monitoring ( ) [static]
```

[DEPRECATED] See: **NDDSUtilityHeapMonitoring::enable** (p. 2114).

9.354.2.5 disable_heap_monitoring()

```
static void NDDSUtility::disable_heap_monitoring ( ) [static]
```

[DEPRECATED] See: **NDDSUtilityHeapMonitoring::disable** (p. 2114).

9.354.2.6 pause_heap_monitoring()

```
static DDS_Boolean NDDSUtility::pause_heap_monitoring ( ) [static]
```

[DEPRECATED] See: **NDDSUtilityHeapMonitoring::pause** (p. 2115).

9.354.2.7 resume_heap_monitoring()

```
static DDS_Boolean NDDSUtility::resume_heap_monitoring ( ) [static]
```

[DEPRECATED] See: **NDDSUtilityHeapMonitoring::resume** (p. 2115)

9.354.2.8 take_heap_snapshot()

```
static DDS_Boolean NDDSUtility::take_heap_snapshot (
            const char * filename,
             DDS_Boolean print_details ) [static]
```

[DEPRECATED] See: **NDDSUtilityHeapMonitoring::take_heap_snapshot** (p. 2115).

9.355 NDDSUtilityHeapMonitoring Class Reference

Heap Monitoring APIs.

Static Public Member Functions

- static **DDS_Boolean** **enable** ()

  Starts monitoring the heap memory used by RTI Connext.
- static **DDS_Boolean** **enable** (const NDDS_Utility_HeapMonitoringParams_t &params)

  Starts monitoring the heap memory used by RTI Connext.
- static void **disable** ()

  Stops monitoring the heap memory used by RTI Connext.
- static **DDS_Boolean** **pause** ()

  Pauses heap monitoring.
- static **DDS_Boolean** **resume** ()

  Resumes heap monitoring.
- static **DDS_Boolean** **take_heap_snapshot** (const char ∗filename, **DDS_Boolean** print_details)

  Saves the current heap memory usage in a file.

9.355.1 Detailed Description

Heap Monitoring APIs.

9.355.2 Member Function Documentation

9.355.2.1 enable() [1/2]

```
static DDS_Boolean NDDSUtilityHeapMonitoring::enable ( ) [static]
```

Starts monitoring the heap memory used by RTI Connext.

This function must be called before any other function in the RTI Connext library is called.

Once heap monitoring is enabled, you can take heap snapshots by using **NDDSUtilityHeapMonitoring::take_heap_snapshot** (p. 2115).

Use this method only for debugging purposes, since it may introduce a significant performance impact.

MT Safety:

UNSAFE. It is not safe to call this method while another thread may be simultaneously calling another heap-related method, including this one.

Returns

DDS_BOOLEAN_TRUE (p. 513) if success. Otherwise, **DDS_BOOLEAN_FALSE** (p. 513)

See also

NDDSUtilityHeapMonitoring::disable (p. 2114)

9.355.2.2 enable() [2/2]

```
static DDS_Boolean NDDSUtilityHeapMonitoring::enable (
            const NDDS_Utility_HeapMonitoringParams_t & params ) [static]
```

Starts monitoring the heap memory used by RTI Connext.

Perfoms the same function as **NDDSUtilityHeapMonitoring::enable** (p. 2114) except that it also provides the values in params. Those values will set the format used in the snapshot **NDDSUtilityHeapMonitoring::take_heap_snapshot** (p. 2115).

Returns

DDS_BOOLEAN_TRUE (p. 513) if success. Otherwise, **DDS_BOOLEAN_FALSE** (p. 513)

See also

NDDSUtilityHeapMonitoring::disable (p. 2114)

9.355.2.3 disable()

```
static void NDDSUtilityHeapMonitoring::disable ( )  [static]
```

Stops monitoring the heap memory used by RTI Connext.

This method must be the last method called from RTI Connext.

See also

NDDSUtilityHeapMonitoring::enable (p. 2114)

9.355.2.4 pause()

```
static  DDS_Boolean NDDSUtilityHeapMonitoring::pause ( )  [static]
```

Pauses heap monitoring.

New memory allocations will not be monitored and they will not appear in the snapshot generated by **NDDSUtility↩ HeapMonitoring::take_heap_snapshot** (p. 2115).

Returns

DDS_BOOLEAN_TRUE (p. 513) if success. Otherwise, **DDS_BOOLEAN_FALSE** (p. 513)

See also

NDDSUtilityHeapMonitoring::resume (p. 2115)

9.355.2.5 resume()

```
static  DDS_Boolean NDDSUtilityHeapMonitoring::resume ( )  [static]
```

Resumes heap monitoring.

Returns

DDS_BOOLEAN_TRUE (p. 513) if success. Otherwise, **DDS_BOOLEAN_FALSE** (p. 513)

See also

NDDSUtilityHeapMonitoring::pause (p. 2115)

9.355.2.6 take_heap_snapshot()

```
static  DDS_Boolean NDDSUtilityHeapMonitoring::take_heap_snapshot (
             const char * filename,
              DDS_Boolean print_details )  [static]
```

Saves the current heap memory usage in a file.

After **NDDSUtilityHeapMonitoring::enable** (p. 2114) is called, you may invoke this method periodically to save the current heap memory usage to a file.

By comparing two snapshots, you can tell if new memory has been allocated and in many cases where. This is why this operation can be used to debug unexpected memory growth.

The format of a snapshot is as follows:

First, there is a memory usage summary like this:

```
<P>
                    Product Version: NDDSCORE_BUILD_6.0.0.0_20200316T123411Z_RTI_ENG

                    Process virtual memory: 2552352768

                    Process physical memory: 16187392

                    Current application heap usage: 10532131

                    Approximate total heap usage: 203331110

                    High watermark: 10532131

                    Alloc count: 17634

                    Free count: 3518
```

- Process virtual memory: The amount of virtual memory in bytes taken by the process. This memory includes RTI Connext and non-RTI Connext memory.
- Process virtual memory: The amount of physical memory in bytes taken by the process.
- Current application heap usage: The amount of heap memory in bytes used by the middleware. For Java and .NET APIs, this memory only accounts for unmanaged RTI Connext memory, not memory living in the managed heap. This value does not include overhead memory allocations that are used by the Heap Monitoring utility. It therefore provides the heap usage that is used when Heap Monitoring is disabled and does not reflect the actual amount of memory that has been allocated by the middleware. That value is accounted for in 'Approximate total heap usage'.
- Approximate total heap usage: The amount of heap memory in bytes used by the middleware, including overhead allocations from the Heap Monitoring utility. When the Heap Monitoring utility is enabled, every allocation has an additional overhead number of bytes allocated so that the middleware can keep track of the meta-data that is output in the heap snapshots. This overhead is not accounted for in the 'Current application heap usage' summary field, but is included in this field. For Java and .NET APIs, this memory only accounts for unmanaged RTI Connext memory, not memory living in the managed heap.
- High watermark: The maximum amount of heap usage by RTI Connext since **NDDSUtilityHeapMonitoring::enable** (p. 2114) was invoked.
- Alloc count: The number of invocations to malloc, realloc, or calloc operations done by RTI Connext.
- Free count: The number of invocations to the free operation done by RTI Connext.

After the previous summary, and only if you set the parameter print_details to **DDS_BOOLEAN_TRUE** (p. 513), the method will print the details of every single outstanding heap allocation done by RTI Connext. For example:

```
<P>
                block_id, timestamp, block_size, alloc_method_name, type_name, pool_alloc, pool_buffer_size,
    pool_buffer_count, topic_name, function_name, activity_context

                23087, 1586943520, 16, RTIOsapiHeap_allocateArray, struct RTIEncapsulationInfo, MALLOC, 0,
    0, PRESServiceRequest, PRESWriterHistoryDriver_new,
    "0X101175A,0X76DD63D7,0X984377BC:0X1C1{Name=ShapeTypeParticipant,Domain=110}|CREATE
    Participant|ENABLE|:0X80000088{Entity=Pu,Domain=110}|CREATE Writer WITH TOPIC PRESServiceRequest"
```

- block_id: Block ID of the allocation. This number increases with every allocation.
- timestamp: Timestamp in UTC seconds corresponding to the time where the allocation was done.
- block_size: The number of bytes allocated.
- alloc_method_name: The allocation RTI Connext method name.
- type_name: The allocation typename.
- pool_alloc: Indicates if the heap allocation is a RTI Connext pool allocation (POOL) or a regular allocation (MALLOC).
- pool_buffer_size: For pool allocations, this number indicates the size of the elements in the pool in number of bytes. block_size is equal to (pool_buffer_size ∗ pool_buffer_count).
- pool_buffer_count: For pool allocations, this number indicates the number of buffers allocated for the pool. block↩ _size is equal to (pool_buffer_size ∗ pool_buffer_count).
- topic_name: The topic name associated with the allocation or 'n/a' if it is not available.
- function_name: function name associated with the allocation or 'n/a' if it is not available.
- activity_context: **Activity Context** (p. 729)

Parameters

filename	<<***in***>> *(p. 433)*. Name of file in which to store the snapshot.
print_details	<<***in***>> *(p. 433)*. Indicates if the snapshot will contain only the memory usage summary or the details of the individual allocations.

Returns

DDS_BOOLEAN_TRUE (p. 513) if success. Otherwise, **DDS_BOOLEAN_FALSE** (p. 513)

9.356 NDDSUtilityNetworkCapture Class Reference

Network Capture APIs.

Static Public Member Functions

- static **DDS_Boolean enable** ()

Enable Network Capture.

- static **DDS_Boolean disable** ()

  Disable Network Capture.

- static **DDS_Boolean set_default_params** (const **NDDS_Utility_NetworkCaptureParams_t** &params)

  Set the default Network Capture parameters.

- static **DDS_Boolean start** (const char *filename)

  Start capturing traffic for all DomainParticipants, with the default parameters.

- static **DDS_Boolean start** (**DDSDomainParticipant** *participant, const char *filename)

  Start capturing traffic for a DomainParticipant, with the default parameters.

- static **DDS_Boolean start** (const char *filename, const **NDDS_Utility_NetworkCaptureParams_t** &params)

  Start capturing traffic for all DomainParticipants, with the provided parameters.

- static **DDS_Boolean start** (**DDSDomainParticipant** *participant, const char *filename, const **NDDS_Utility_NetworkCaptureParams_t** &params)

  Start capturing traffic for a DomainParticipant, with the provided parameters.

- static **DDS_Boolean stop** ()

  Stop capturing traffic for all participants.

- static **DDS_Boolean stop** (**DDSDomainParticipant** *participant)

  Stop capturing traffic for a DomainParticipant.

- static **DDS_Boolean pause** ()

  Pause capturing traffic for all DomainParticipants.

- static **DDS_Boolean pause** (**DDSDomainParticipant** *participant)

  Pause capturing traffic for a DomainParticipant.

- static **DDS_Boolean resume** ()

  Resume capturing traffic for all DomainParticipants.

- static **DDS_Boolean resume** (**DDSDomainParticipant** *participant)

  Resume capturing traffic for a DomainParticipant.

9.356.1 Detailed Description

Network Capture APIs.

9.356.2 Member Function Documentation

9.356.2.1 enable()

```
static DDS_Boolean NDDSUtilityNetworkCapture::enable ( ) [static]
```

Enable Network Capture.

This method must be called before any other Network Capture method. It must also be called before creating the participants for which we want to capture traffic.

Use this method only for debugging purposes, since it may introduce a significant performance impact.

Returns

DDS_BOOLEAN_TRUE (p. 513) if success. Otherwise, **DDS_BOOLEAN_FALSE** (p. 513)

MT Safety:

UNSAFE. It is not safe to call this method while another thread may be simultaneously calling another Network Caputre related method, including this one.

See also

NDDSUtilityNetworkCapture::disable (p. 2119)

9.356.2.2 disable()

```
static DDS_Boolean NDDSUtilityNetworkCapture::disable ( ) [static]
```

Disable Network Capture.

This method must be the last Network Capture method to be called. It must also be called after deleting the participants for which we captured traffic. Disabling Network Capture without stopping it first is not ok!

Returns

DDS_BOOLEAN_TRUE (p. 513) if success. Otherwise, **DDS_BOOLEAN_FALSE** (p. 513)

MT Safety:

UNSAFE. It is not safe to call this method while another thread may be simultaneously calling another Network Capture related method, including this one.

See also

NDDSUtilityNetworkCapture::enable (p. 2118)

9.356.2.3 set_default_params()

```
static DDS_Boolean NDDSUtilityNetworkCapture::set_default_params (
            const NDDS_Utility_NetworkCaptureParams_t & params ) [static]
```

Set the default Network Capture parameters.

The default parameters are used when Network Capture is started without parameters, i.e., **NDDSUtilityNetwork↩ Capture::start** (p. 2120).

Precondition

This method requires first enabling Network Capture. See **NDDSUtilityNetworkCapture::enable** (p. 2118).

Parameters

params	<<***in***>> *(*p. *433).* Configuration parameters that we want to set as defaults.

Returns

DDS_BOOLEAN_TRUE (p. 513) if success. Otherwise, **DDS_BOOLEAN_FALSE** (p. 513)

See also

NDDSUtilityNetworkCapture::start (p. 2120)

NDDSUtilityNetworkCapture::start(DDSDomainParticipant ∗, const char ∗) (p. 2120)

9.356.2.4 start() [1/4]

```
static  DDS_Boolean NDDSUtilityNetworkCapture::start (
             const char * filename )  [static]
```

Start capturing traffic for all DomainParticipants, with the default parameters.

Precondition

This method requires first enabling Network Capture. See **NDDSUtilityNetworkCapture::enable** (p. 2118).

Parameters

filename	<<***in***>> *(*p. *433).* The name of the output capture file will be based on this input parameter.

In particular, the name for the capture file is the concatenation of the `filename` input parameter, the `"_GUID-"` string followed by the decimal representation of bytes 8-11 of the DomainParticipant's GUID, and the file extension (`".pcap"`).

Returns

DDS_BOOLEAN_TRUE (p. 513) if success. Otherwise, **DDS_BOOLEAN_FALSE** (p. 513)

See also

NDDSUtilityNetworkCapture::stop (p. 2123)

NDDSUtilityNetworkCapture::start(DDSDomainParticipant ∗, const char ∗) (p. 2120)

9.356.2.5 start() [2/4]

```
static DDS_Boolean NDDSUtilityNetworkCapture::start (
            DDSDomainParticipant * participant,
            const char * filename ) [static]
```

Start capturing traffic for a DomainParticipant, with the default parameters.

Precondition

This method requires first enabling Network Capture. See **NDDSUtilityNetworkCapture::enable** (p. 2118).

Parameters

participant	<<***in***>> *(p. 433).* DomainParticipant for which we want to capture traffic.
filename	<<***in***>> *(p. 433).* The name of the output capture file will be based on this input parameter.

In particular, the name for the capture file is the concatenation of the `filename` input parameter, and the file extension (`".pcap"`).

Returns

DDS_BOOLEAN_TRUE (p. 513) if success. Otherwise, **DDS_BOOLEAN_FALSE** (p. 513)

See also

NDDSUtilityNetworkCapture::stop(DDSDomainParticipant ∗) (p. 2123)

NDDSUtilityNetworkCapture::start(DDSDomainParticipant ∗, const char ∗, const NDDS_Utility_Network↩ CaptureParams_t&) (p. 2122)

NDDSUtilityNetworkCapture::enable (p. 2118)

9.356.2.6 start() [3/4]

```
static DDS_Boolean NDDSUtilityNetworkCapture::start (
            const char * filename,
            const NDDS_Utility_NetworkCaptureParams_t & params ) [static]
```

Start capturing traffic for all DomainParticipants, with the provided parameters.

Precondition

This method requires first enabling Network Capture. See **NDDSUtilityNetworkCapture::enable** (p. 2118).

Perfoms the same function as **NDDSUtilityNetworkCapture::start** (p. 2120) except that it uses the provided parameters, instead of the default ones.

Parameters

filename	<<***in***>> *(p. 433).* The name of the output capture file will be based on this input parameter.

In particular, the name for the capture file is the concatenation of the `filename` input parameter, the `"_GUID-"` string followed by the decimal representation of bytes 8-11 of the DomainParticipant's GUID, and the file extension (`".pcap"`).

Parameters

params	<<***in***>> *(p. 433).* Configuration parameters for the capture.

Returns

DDS_BOOLEAN_TRUE (p. 513) if success. Otherwise, **DDS_BOOLEAN_FALSE** (p. 513)

See also

NDDSUtilityNetworkCapture::stop (p. 2123)

NDDSUtilityNetworkCapture::start(DDSDomainParticipant ∗, const char ∗, const NDDS_Utility_Network↩ CaptureParams_t&) (p. 2122)

9.356.2.7 start() [4/4]

```
static  DDS_Boolean NDDSUtilityNetworkCapture::start (
              DDSDomainParticipant * participant,
             const char * filename,
             const  NDDS_Utility_NetworkCaptureParams_t & params )  [static]
```

Start capturing traffic for a DomainParticipant, with the provided parameters.

Precondition

This method requires enabling first Network Capture. See **NDDSUtilityNetworkCapture::enable** (p. 2118).

Parameters

participant	<<***in***>> *(p. 433).* DomainParticipant for which we want to capture traffic.
filename	<<***in***>> *(p. 433).* The name of the output capture file will be based on this input parameter.

In particular, the name for the capture file is the concatenation of the `filename` input parameter, and the file extension (`".pcap"`).

Parameters

params	<<***in***>> (p. *433*). Parameters for configuring the capture.

Returns

DDS_BOOLEAN_TRUE (p. 513) if success. Otherwise, **DDS_BOOLEAN_FALSE** (p. 513)

See also

NDDSUtilityNetworkCapture::stop(DDSDomainParticipant ∗) (p. 2123)

NDDSUtilityNetworkCapture::start(DDSDomainParticipant ∗, const char ∗) (p. 2120)

9.356.2.8 stop() [1/2]

```
static DDS_Boolean NDDSUtilityNetworkCapture::stop ( ) [static]
```

Stop capturing traffic for all participants.

Precondition

This method requires enabling first Network Capture. See **NDDSUtilityNetworkCapture::enable** (p. 2118).

This method can (and must) be called after **NDDSUtilityNetworkCapture::start** (p. 2120), not **NDDSUtilityNetwork↩ Capture::start(DDSDomainParticipant ∗, const char ∗)** (p. 2120). That is, if we start capturing traffic globally (for all DomainParticipants), we must stop capturing traffic also globally. It is not possible to start capturing traffic for a participant but stop it globally.

It is possible to start capturing globally and then stop capturing for a participant, as long as we eventually stop capturing traffic globally.

We must stop capturing for a participant before deleting it.

Returns

DDS_BOOLEAN_TRUE (p. 513) if success. Otherwise, **DDS_BOOLEAN_FALSE** (p. 513)

See also

NDDSUtilityNetworkCapture::start (p. 2120)

NDDSUtilityNetworkCapture::stop(DDSDomainParticipant ∗) (p. 2123)

9.356.2.9 stop() [2/2]

```
static DDS_Boolean NDDSUtilityNetworkCapture::stop (
            DDSDomainParticipant * participant ) [static]
```

Stop capturing traffic for a DomainParticipant.

Precondition

This method requires first enabling Network Capture. See **NDDSUtilityNetworkCapture::enable** (p. 2118).

Parameters

participant	<<***in***>> *(p. 433).* DomainParticipant for which we want to stop capturing traffic.

Returns

DDS_BOOLEAN_TRUE (p. 513) if success. Otherwise, **DDS_BOOLEAN_FALSE** (p. 513)

See also

NDDSUtilityNetworkCapture::start(DDSDomainParticipant ∗, const char ∗) (p. 2120)

NDDSUtilityNetworkCapture::stop (p. 2123)

9.356.2.10 pause() [1/2]

```
static DDS_Boolean NDDSUtilityNetworkCapture::pause ( ) [static]
```

Pause capturing traffic for all DomainParticipants.

Precondition

This method requires first enabling Network Capture. See **NDDSUtilityNetworkCapture::enable** (p. 2118).

Returns

DDS_BOOLEAN_TRUE (p. 513) if success. Otherwise, **DDS_BOOLEAN_FALSE** (p. 513)

See also

NDDSUtilityNetworkCapture::resume (p. 2125)

NDDSUtilityNetworkCapture::pause(DDSDomainParticipant ∗) (p. 2124)

9.356.2.11 pause() [2/2]

```
static DDS_Boolean NDDSUtilityNetworkCapture::pause (
            DDSDomainParticipant * participant ) [static]
```

Pause capturing traffic for a DomainParticipant.

Precondition

This method requires first enabling Network Capture. See **NDDSUtilityNetworkCapture::enable** (p. 2118).

Parameters

participant	<<***in***>> *(p. 433).* DomainParticipant for which we want to pause capturing traffic.

Returns

DDS_BOOLEAN_TRUE (p. 513) if success. Otherwise, **DDS_BOOLEAN_FALSE** (p. 513)

See also

NDDSUtilityNetworkCapture::resume(DDSDomainParticipant ∗) (p. 2125)

NDDSUtilityNetworkCapture::pause (p. 2124)

9.356.2.12 resume() [1/2]

```
static DDS_Boolean NDDSUtilityNetworkCapture::resume ( ) [static]
```

Resume capturing traffic for all DomainParticipants.

Precondition

This method requires first enabling Network Capture. See **NDDSUtilityNetworkCapture::enable** (p. 2118).

Returns

DDS_BOOLEAN_TRUE (p. 513) if success. Otherwise, **DDS_BOOLEAN_FALSE** (p. 513)

See also

NDDSUtilityNetworkCapture::pause (p. 2124)

NDDSUtilityNetworkCapture::resume(DDSDomainParticipant ∗) (p. 2125)

9.356.2.13 resume() [2/2]

```
static DDS_Boolean NDDSUtilityNetworkCapture::resume (
            DDSDomainParticipant * participant ) [static]
```

Resume capturing traffic for a DomainParticipant.

Precondition

This method requires first enabling Network Capture. See **NDDSUtilityNetworkCapture::enable** (p. 2118).

Parameters

participant	<<***in***>> *(p. 433).* DomainParticipant for which we want to resume capturing traffic.

Returns

DDS_BOOLEAN_TRUE (p. 513) if success. Otherwise, **DDS_BOOLEAN_FALSE** (p. 513)

See also

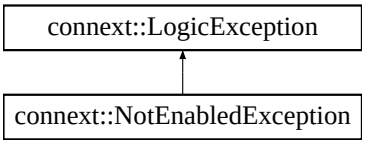
NDDSUtilityNetworkCapture::pause(DDSDomainParticipant ∗) (p. 2124)

NDDSUtilityNetworkCapture::resume (p. 2125)

9.357 connext::NotEnabledException Class Reference

Operation invoked on a **DDSEntity** (p. 1731) that is not yet enabled.

Inheritance diagram for connext::NotEnabledException:



9.357.1 Detailed Description

Operation invoked on a **DDSEntity** (p. 1731) that is not yet enabled.

See also

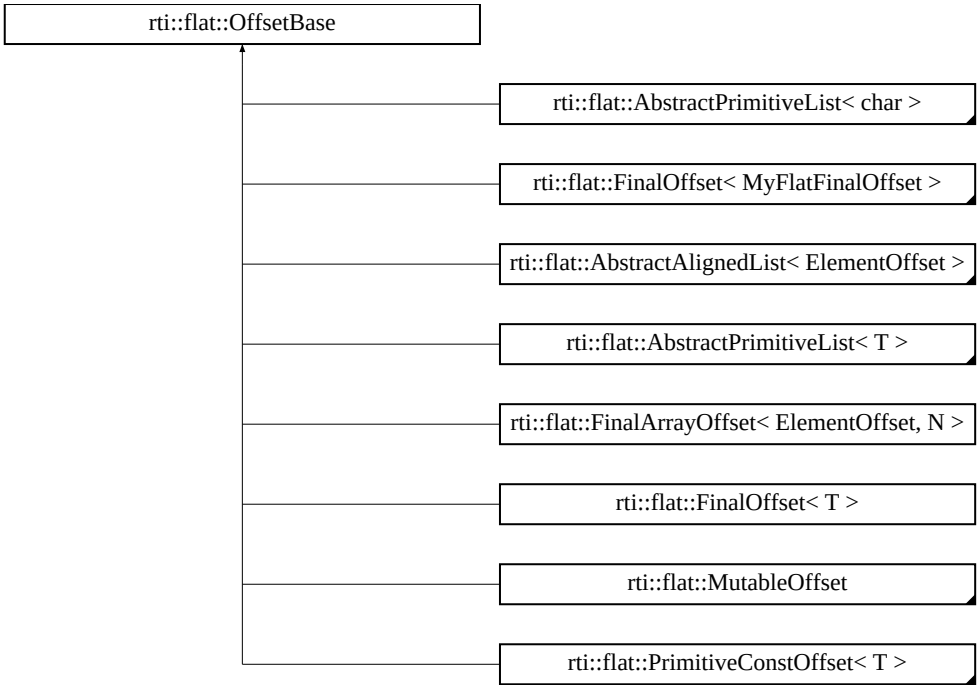
DDS_RETCODE_NOT_ENABLED (p. 533)

9.358 rti::flat::OffsetBase Class Reference

Base class of all Offset types.

`#include <Offset.hpp>`

Inheritance diagram for rti::flat::OffsetBase:



Public Member Functions

- bool **is_null** () const

  Indicates whether this Offset doesn't point to a valid element.
- bool **is_cpp_compatible** () const

  *Indicates whether **rti::flat::plain_cast()** (p. 773) is possible.*
- const unsigned char * **get_buffer** () const

  Gets this member's position in the buffer.
- offset_t **get_buffer_size** () const

  Gets the size, in bytes, of this member in the buffer.

Friends

- bool **operator<** (const **OffsetBase** &s1, const **OffsetBase** &s2)

  Compares two Offsets.
- bool **operator>** (const **OffsetBase** &s1, const **OffsetBase** &s2)

  Compares two Offsets.
- bool **operator<=** (const **OffsetBase** &s1, const **OffsetBase** &s2)

  Compares two Offsets.
- bool **operator>=** (const **OffsetBase** &s1, const **OffsetBase** &s2)

  Compares two Offsets.
- bool **operator==** (const **OffsetBase** &s1, const **OffsetBase** &s2)

  Determines if two offets point to the same position.
- bool **operator!=** (const **OffsetBase** &s1, const **OffsetBase** &s2)

  Determines if two offets point to different positions.

9.358.1 Detailed Description

Base class of all Offset types.

See also

FlatData Offsets (p. 771)

9.358.2 Member Function Documentation

9.358.2.1 is_null()

```
bool rti::flat::OffsetBase::is_null ( ) const  [inline]
```

Indicates whether this Offset doesn't point to a valid element.

See also

Offset Error Management (p. 773)

Referenced by **get_buffer()**.

9.358.2.2 is_cpp_compatible()

```
bool rti::flat::OffsetBase::is_cpp_compatible ( ) const  [inline]
```

Indicates whether **rti::flat::plain_cast()** (p. 773) is possible.

Returns

True only if the data pointed to by this Offset can be **rti::flat::plain_cast()** (p. 773)

See also

rti::flat::plain_cast() (p. 773) for the requirements that a type needs to meet

9.358.2.3 get_buffer()

```
const unsigned char * rti::flat::OffsetBase::get_buffer ( ) const  [inline]
```

Gets this member's position in the buffer.

Note

This function should be used for debugging purposes only. To access the data in this Offset use the Offset accessor methods or, if this type allows it, **rti::flat::plain_cast()** (p. 773).

Returns the position within the **Sample** (p. 2187)'s buffer that this Offset points to.

See also

get_buffer_size() (p. 2129)

References **is_null()**.

Referenced by **rti::flat::StringOffset::get_string()**.

9.358.2.4 get_buffer_size()

```
offset_t rti::flat::OffsetBase::get_buffer_size ( ) const  [inline]
```

Gets the size, in bytes, of this member in the buffer.

Returns the number of bytes that this member comprises after the position returned by get_buffer().

Referenced by **rti::flat::AbstractAlignedList< ElementOffset >::begin()**, and **rti::flat::AbstractAlignedList< ElementOffset >::end()**.

9.358.3 Friends And Related Function Documentation

9.358.3.1 operator<

```
bool operator< (
            const  OffsetBase & s1,
            const  OffsetBase & s2 )  [friend]
```

Compares two Offsets.

Returns

True if s1 points to a position smaller than s2's.

9.358.3.2 operator>

```
bool operator> (
            const  OffsetBase & s1,
            const  OffsetBase & s2 )  [friend]
```

Compares two Offsets.

Returns

True if s1 points to a position greater than s2's.

9.358.3.3 operator<=

```
bool operator<= (
            const  OffsetBase & s1,
            const  OffsetBase & s2 )  [friend]
```

Compares two Offsets.

Returns

True if s1 points to a position equal to or smaller than s2's.

9.358.3.4 operator>=

```
bool operator>= (
            const  OffsetBase & s1,
            const  OffsetBase & s2 )  [friend]
```

Compares two Offsets.

Returns

True if s1 points to a position equal to or greater than s2's.

9.358.3.5 operator==

```
bool operator== (
            const  OffsetBase & s1,
            const  OffsetBase & s2 )  [friend]
```

Determines if two offets point to the same position.

Returns

True if s1 points to the same position as s2.

9.358.3.6 operator"!=

```
bool operator!= (
            const  OffsetBase & s1,
            const  OffsetBase & s2 )  [friend]
```

Determines if two offets point to different positions.

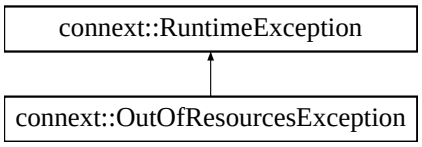
Returns

True if s1 points to a different position than s2.

9.359 connext::OutOfResourcesException Class Reference

RTI Connext ran out of the resources needed to complete the operation.

Inheritance diagram for connext::OutOfResourcesException:



9.359.1 Detailed Description

RTI Connext ran out of the resources needed to complete the operation.

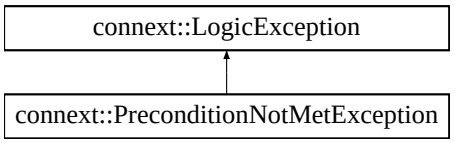
See also

DDS_RETCODE_OUT_OF_RESOURCES (p. 533)

9.360 connext::PreconditionNotMetException Class Reference

A pre-condition for the operation was not met.

Inheritance diagram for connext::PreconditionNotMetException:



9.360.1 Detailed Description

A pre-condition for the operation was not met.

See also

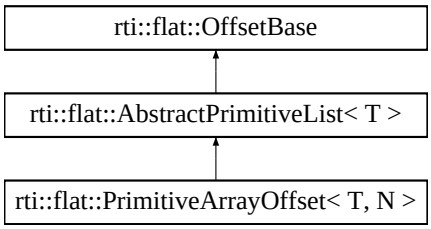
DDS_RETCODE_PRECONDITION_NOT_MET (p. 532)

9.361 rti::flat::PrimitiveArrayOffset< T, N > Class Template Reference

Offset to an array of primitive elements.

`#include <SequenceOffsets.hpp>`

Inheritance diagram for rti::flat::PrimitiveArrayOffset< T, N >:



Public Member Functions

- unsigned int **element_count** () const

  Returns the number of elements, N.

9.361.1 Detailed Description

template<typename T, unsigned int N>
class rti::flat::PrimitiveArrayOffset< T, N >

Offset to an array of primitive elements.

Template Parameters

T	The primitive type
N	The array bound. For multidimensional arrays, N is the product of each dimension bound.

A **PrimitiveArrayOffset** (p. 2132) may meet the requirements to be cast to an array of the equivalent plain C++ element type (see **rti::flat::plain_cast()** (p. 773)).

9.361.2 Member Function Documentation

9.361.2.1 element_count()

```
template<typename T , unsigned int N>
unsigned int  rti::flat::PrimitiveArrayOffset< T, N >::element_count ( ) const  [inline]
```

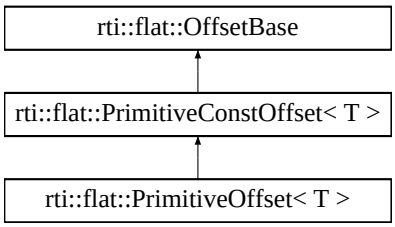
Returns the number of elements, N.

9.362 rti::flat::PrimitiveConstOffset< T > Struct Template Reference

A const Offset to an optional primitive member.

```
#include <Offset.hpp>
```

Inheritance diagram for rti::flat::PrimitiveConstOffset< T >:



Public Member Functions

- T **get** () const

  Gets the value of this primitive member.

9.362.1 Detailed Description

template<typename T>
struct rti::flat::PrimitiveConstOffset< T >

A const Offset to an optional primitive member.

Read-only version of **PrimitiveOffset** (p. 2134).

If !is_null(), the value can be retrieved with **get()** (p. 2134).

9.362.2 Member Function Documentation

9.362.2.1 get()

```
template<typename T >
T  rti::flat::PrimitiveConstOffset< T >::get ( ) const  [inline]
```

Gets the value of this primitive member.

Precondition

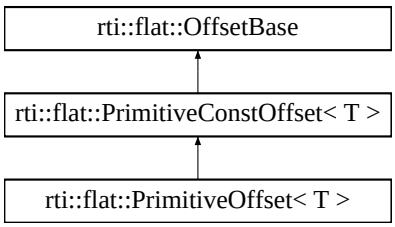
!is_null()

9.363 rti::flat::PrimitiveOffset< T > Struct Template Reference

An Offset to an optional primitive member.

`#include <Offset.hpp>`

Inheritance diagram for rti::flat::PrimitiveOffset< T >:



Public Member Functions

- bool **set** (T value)

  Sets a value for this primitive member.

9.363.1 Detailed Description

template<typename T>
struct rti::flat::PrimitiveOffset< T >

An Offset to an optional primitive member.

Template Parameters

T	The primitive type

Non-optional primitive members are accessed directly using its container type's methods; however, since an optional member may not exist, this Offset is returned instead. The only purpose of the type **PrimitiveOffset** (p. 2134) is to provide a way to check for the member's existence.

If !is_null(), the value can be retrieved with **get()** (p. 2134) or set with **set()** (p. 2135).

See also

MyFlatMutableOffset::my_optional_primitive (p. 2027) vs. **MyFlatMutableOffset::my_primitive** (p. 2027)

9.363.2 Member Function Documentation

9.363.2.1 set()

```
template<typename T >
bool  rti::flat::PrimitiveOffset< T >::set (
             T value )  [inline]
```

Sets a value for this primitive member.

Precondition

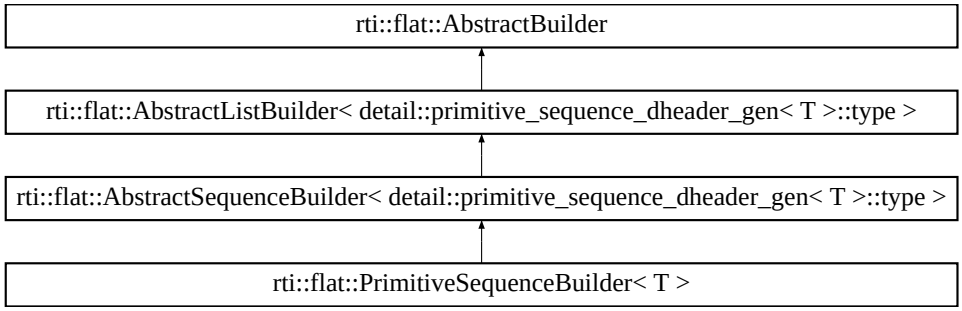
!is_null()

9.364 rti::flat::PrimitiveSequenceBuilder< T > Class Template Reference

Builds a sequence of primitive members.

```
#include <SequenceBuilders.hpp>
```

Inheritance diagram for rti::flat::PrimitiveSequenceBuilder< T >:



Public Member Functions

- **PrimitiveSequenceBuilder** & **add_next** (T value)

  Adds the next element.
- **PrimitiveSequenceBuilder** & **add_n** (const T *array, unsigned int count)

  Adds all the elements in an array.
- **PrimitiveSequenceBuilder** & **add_n** (unsigned int count, T value)

  Adds a number of elements with the same value.
- **PrimitiveSequenceBuilder** & **add_n** (unsigned int count)

  Adds a number of uninitialized elements.
- **Offset finish** ()

  Finishes building the sequence.

Additional Inherited Members

9.364.1 Detailed Description

template<typename T>
class rti::flat::PrimitiveSequenceBuilder< T >

Builds a sequence of primitive members.

The elements can be added one by one with add_next() or all at once with **add_n()** (p. 2136).

9.364.2 Member Function Documentation

9.364.2.1 add_next()

```
template<typename T >
PrimitiveSequenceBuilder & rti::flat::PrimitiveSequenceBuilder< T >::add_next (
            T value ) [inline]
```

Adds the next element.

Parameters

value	The primitive element to add

9.364.2.2 add_n() [1/3]

```
template<typename T >
PrimitiveSequenceBuilder &  rti::flat::PrimitiveSequenceBuilder< T >::add_n (
            const T * array,
            unsigned int count )  [inline]
```

Adds all the elements in an array.

Parameters

array	The array containing the values to add
count	The size of the array

Precondition

array must not be NULL

9.364.2.3 add_n() [2/3]

```
template<typename T >
PrimitiveSequenceBuilder &  rti::flat::PrimitiveSequenceBuilder< T >::add_n (
            unsigned int count,
            T value )  [inline]
```

Adds a number of elements with the same value.

This overload initializes all the elements to a value.

Note

This operation is O(`count`). The other overloads, **add_n(unsigned int)** (p. 2137) and **add_n(const T*, unsigned int)** (p. 2136), are more efficient

Parameters

count	The number of elements to add
value	The value to set for each element

9.364.2.4 add_n() [3/3]

```
template<typename T >
```

```
PrimitiveSequenceBuilder &  rti::flat::PrimitiveSequenceBuilder< T >::add_n (
            unsigned int count )  [inline]
```

Adds a number of uninitialized elements.

This operation is O(1), since it leaves the elements uninitialized. They can be initialized using the Offset returned by **finish()** (p. 2138) and **rti::flat::plain_cast()** (p. 773). For example:

```
MyFlatMutableBuilder builder = ...;
auto seq_builder = builder.build_my_primitive_seq();
seq_builder.add_n(1000);
auto seq_offset = seq_builder.finish();
int32_t *elements = rti::flat::plain_cast(seq_offset);
for (int i = 0; i < 1000; i++) {
    elements[i] = ...;
}
```

Parameters

count	The number of elements to add

9.364.2.5 finish()

```
template<typename T >
Offset  rti::flat::PrimitiveSequenceBuilder< T >::finish ( )  [inline]
```

Finishes building the sequence.

Returns

An Offset to the member that has been built.

See also

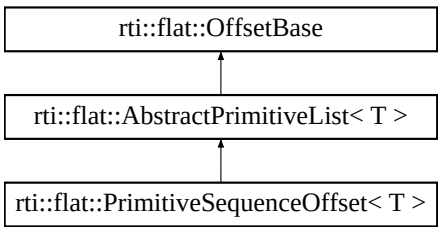
discard() (p. 788)

9.365 rti::flat::PrimitiveSequenceOffset< T > Class Template Reference

Offset to a sequence of primitive elements.

```
#include <SequenceOffsets.hpp>
```

Inheritance diagram for rti::flat::PrimitiveSequenceOffset< T >:



Public Member Functions

- unsigned int **element_count** () const

  Returns the number of elements.

9.365.1 Detailed Description

template<typename T>
class rti::flat::PrimitiveSequenceOffset< T >

Offset to a sequence of primitive elements.

Template Parameters

T	The primitive type

9.365.2 Member Function Documentation

9.365.2.1 element_count()

```
template<typename T >
unsigned int  rti::flat::PrimitiveSequenceOffset< T >::element_count ( ) const  [inline]
```

Returns the number of elements.

Referenced by **rti::flat::StringOffset::element_count()**.

9.366 connext::Replier< TReq, TRep > Class Template Reference

Allows receiving requests and sending replies.

Public Types

- typedef TReq **Request**

  The request type.
- typedef TRep **Reply**

  The reply type.
- typedef dds_type_traits< TRep >::DataWriter **ReplyDataWriter**

  The typed DataWriter for type `TRep`.
- typedef dds_type_traits< TReq >::DataReader **RequestDataReader**

  The typed DataReader for type `TReq`.

Public Member Functions

- **Replier** (**DDSDomainParticipant** *participant, const std::string &service_name)

  *Creates a **Replier** (p. 2139) with the minimum set of parameters.*
- **Replier** (const **ReplierParams**< TReq, TRep > &params)

  *Creates a **Replier** (p. 2139) with parameters.*
- virtual ~**Replier** ()

  *Releases the internal entities created by this **Replier** (p. 2139).*
- template<typename URep >
  void **send_reply** (const URep &reply, const **SampleIdentity_t** &related_request_id)

  Sends a reply for a previous request.
- template<typename URep >
  void **send_reply** (**WriteSample**< URep > &reply, const **SampleIdentity_t** &related_request_id)

  Sends a reply for a previous request.
- template<typename URep >
  void **send_reply** (**WriteSampleRef**< URep > &reply, const **SampleIdentity_t** &related_request_id)

  *Sends a reply for a previous request using a **SampleRef** (p. 2191).*
- bool **receive_request** (**connext::Sample**< TReq > &request, const **DDS_Duration_t** &max_wait)

  *Waits for a request and copies its contents into a **Sample** (p. 2183).*
- bool **receive_request** (**connext::SampleRef**< TReq > request, const **DDS_Duration_t** &max_wait)

  *Waits for a request and copies its contents into a **SampleRef** (p. 2191).*
- **LoanedSamplesType** **receive_requests** (const **DDS_Duration_t** &max_wait)

  Waits for multiple requests and provides a loaned container to access them.
- **LoanedSamplesType** **receive_requests** (int min_request_count, int max_request_count, const **DDS_↩ Duration_t** &max_wait)

  Waits for multiple requests and provides a loaned container to access them.
- bool **wait_for_requests** (const **DDS_Duration_t** &max_wait)

  Waits for requests.
- bool **wait_for_requests** (int min_count, const **DDS_Duration_t** &max_wait)

  Waits for requests.
- bool **take_request** (**connext::Sample**< TReq > &request)

  *Copies the contents of a request into a **Sample** (p. 2183).*
- bool **take_request** (**connext::SampleRef**< TReq > request)

  *Copies the contents of a request into a **SampleRef** (p. 2191).*
- **LoanedSamplesType** **take_requests** (int max_samples= **DDS_LENGTH_UNLIMITED**)

  Provides a loaned container to access the existing requests.
- bool **read_request** (**connext::Sample**< TReq > &request)

  *Copies the contents of a request into a **Sample** (p. 2183).*
- bool **read_request** (**connext::SampleRef**< TReq > request)

  *Copies the contents of a request into a **SampleRef** (p. 2191).*
- **LoanedSamplesType** **read_requests** (int max_samples= **DDS_LENGTH_UNLIMITED**)

  Provides a loaned container to access the existing requests.
- **RequestDataReader** * **get_request_datareader** () const

  *Retrieves the underlying **DDSDataReader** (p. 1556).*
- **ReplyDataWriter** * **get_reply_datawriter** () const

  *Retrieves the underlying **DDSDataWriter** (p. 1589).*

9.366.1 Detailed Description

template<typename TReq, typename TRep>
class connext::Replier< TReq, TRep >

Allows receiving requests and sending replies.

A **Replier** (p. 2139) is an entity with two associated **topics** (p. 63): a request topic and a reply topic. It can receive requests by subscribing to the request topic and can send replies to those requests by publishing the reply topic.

Valid types for these topics (`TReq` and `TRep`) are: those generated by rtiddsgen, the **DDS built-in types** (p. 95), and **DDS_DynamicData** (p. 1013). See **Creating a Replier** (p. 425).

A **Replier** (p. 2139) has four main types of operations:

- Waiting for requests to be received from the middleware
- Getting those requests
- Receiving requests (a convenience operation that is a combination of waiting and getting in a single operation)
- Sending a reply for a previously received request (i.e., publishing a reply sample on the reply topic with special meta-data so that the orignal **Requester** (p. 2157) can identify it)

For multi-reply scenarios in which a **connext::Replier** (p. 2139) generates more than one reply for a request, the **connext::Replier** (p. 2139) should mark all the intermediate replies (all but the last) using the **DDS_INTERMEDIATE_REPLY_SEQUENCE_SAMPLE** (p. 673) flag in **DDS_WriteParams_t::flag** (p. 1522).

Much like a **Requester** (p. 2157), a **Replier** (p. 2139) has an associated **DDSDomainParticipant** (p. 1619), which can be shared with other Repliers or RTI Connext routines. All the other entities required for the request-reply interaction, including a **DDSDataWriter** (p. 1589) for writing replies and a **DDSDataReader** (p. 1556) for reading requests, are automatically created when the **Replier** (p. 2139) is constructed.

Quality of Service for the underlying DataWriter and DataReader can be configured (see **connext::RequesterParams::qos_profile** (p. 2180)). By default, they are created with **DDS_RELIABLE_RELIABILITY_QOS** (p. 633). The exact default configuration is described here: **Configuring Request-Reply QoS profiles** (p. 427)

There are several ways to use a **Replier** (p. 2139):

- A thread **receives** (p. 2147) requests and then dispatches them. If the computation of a reply is a simple operation, consider using a **connext::SimpleReplier** (p. 2206) instead of a **Replier** (p. 2139).
- Polling without waiting, using **take_requests(int)** (p. 2149) directly.
- Using a **connext::ReplierListener** (p. 2151) to get notified and **get** (p. 2149) the requests within the callback.

Template Parameters

TReq	The data type for the request topic
TRep	The data type for the reply topic

See also

connext::Requester (p. 2157)

Request-Reply Examples (p. 419)

Basic Replier example (p. 425)

9.366.2 Member Typedef Documentation

9.366.2.1 Request

```
template<typename TReq , typename TRep >
typedef TReq  connext::Replier< TReq, TRep >::Request
```

The request type.

9.366.2.2 Reply

```
template<typename TReq , typename TRep >
typedef TRep  connext::Replier< TReq, TRep >::Reply
```

The reply type.

9.366.2.3 ReplyDataWriter

```
template<typename TReq , typename TRep >
typedef dds_type_traits<TRep>::DataWriter  connext::Replier< TReq, TRep >::ReplyDataWriter
```

The typed DataWriter for type `TRep`.

9.366.2.4 RequestDataReader

```
template<typename TReq , typename TRep >
typedef dds_type_traits<TReq>::DataReader  connext::Replier< TReq, TRep >::RequestDataReader
```

The typed DataReader for type `TReq`.

9.366.3 Constructor & Destructor Documentation

9.366.3.1 Replier() [1/2]

```
template<typename TReq , typename TRep >
connext::Replier< TReq, TRep >::Replier (
             DDSDomainParticipant * participant,
            const std::string & service_name )
```

Creates a **Replier** (p. 2139) with the minimum set of parameters.

Parameters

participant	The DomainParticipant that this **Replier** (p. 2139) uses to join a domain.
service_name	The service name. See **connext::ReplierParams::service_name** (p. 2154)

Exceptions

One	of the **RTI Connext Exceptions** (p. 39)

9.366.3.2 Replier() [2/2]

```
template<typename TReq , typename TRep >
connext::Replier< TReq, TRep >::Replier (
            const  ReplierParams< TReq, TRep > & params )  [explicit]
```

Creates a **Replier** (p. 2139) with parameters.

Parameters

params	All the parameters that configure this **Replier** (p. 2139)

Exceptions

One	of the **RTI Connext Exceptions** (p. 39)

See also

connext::ReplierParams (p. 2152)

Creating a Replier (p. 425)

9.366.3.3 ~Replier()

```
template<typename TReq , typename TRep >
connext::Replier< TReq, TRep >::~ Replier  [virtual]
```

Releases the internal entities created by this **Replier** (p. 2139).

Among other internal resources, it deletes the **Replier** (p. 2139)'s underlying DataReader and DataWriter.

See also

DDSSubscriber::delete_datareader (p. 1870)

DDSPublisher::delete_datawriter (p. 1829)

9.366.4 Member Function Documentation

9.366.4.1 send_reply() [1/3]

```
template<typename TReq , typename TRep >
template<typename URep >
void  connext::Replier< TReq, TRep >::send_reply (
            const URep & reply,
            const  SampleIdentity_t & related_request_id )
```

Sends a reply for a previous request.

The related request identity can be retrieved from an existing request sample (**connext::Sample** (p. 2183)).

Parameters

reply	The reply to be sent.
related_request_id	The identity of a previously received request

Exceptions

One	of the **RTI Connext Exceptions** (p. 39)

See also

connext::Sample::identity() (p. 2185)

receive_request(Sample<TReq>&, const Duration_t&) (p. 2146)

receive_requests(int, int, const Duration_t&) (p. 2147)

take_request(Sample<TReq>&) (p. 2149)

take_requests(int) (p. 2149)

Basic Replier example (p. 425)

9.366.4.2 send_reply() [2/3]

```
template<typename TReq , typename TRep >
template<typename URep >
void  connext::Replier< TReq, TRep >::send_reply (
             WriteSample< URep > & reply,
            const  SampleIdentity_t & related_request_id )
```

Sends a reply for a previous request.

Allows you to set custom parameters for writing a reply.

Contrary to the **connext::Requester** (p. 2157), where retrieving the sample identity for correlation is common, on the **Replier** (p. 2139) side using a **WriteSample** (p. 2219) is only necessary when the default write parameters need to be overridden, and **send_reply(const URep&, const SampleIdentity_t&)** (p. 2145) may be used if that is not necessary.

One reason to override the default write parameters is a multi-reply scenario in which a **connext::Replier** (p. 2139) generates more than one reply for a request. In this case, all the intermediate replies (all but the last) should be marked with the **DDS_INTERMEDIATE_REPLY_SEQUENCE_SAMPLE** (p. 673) flag in **DDS_WriteParams_t::flag** (p. 1522) within **connext::WriteSample::info()** (p. 2222).

A **connext::Requester** (p. 2157) can detect if a reply is the last reply of a sequence of replies by checking that the flag **DDS_INTERMEDIATE_REPLY_SEQUENCE_SAMPLE** (p. 673) is not set in **DDS_SampleInfo::flag** (p. 1353) within **connext::Sample::info()** (p. 2186).

See also

send_reply(const URep&, const SampleIdentity_t&) (p. 2145)

connext::WriteSample (p. 2219)

9.366.4.3 send_reply() [3/3]

```
template<typename TReq , typename TRep >
template<typename URep >
void  connext::Replier< TReq, TRep >::send_reply (
              WriteSampleRef< URep > & reply,
             const  SampleIdentity_t & related_request_id )
```

Sends a reply for a previous request using a **SampleRef** (p. 2191).

See also

connext::WriteSampleRef (p. 2223)

connext::Requester::send_request(WriteSampleRef<UReq>&) (p. 2164)

connext::Replier::send_reply(WriteSample<URep>&, const SampleIdentity_t&) (p. 2145)

Basic Requester example using SampleRef (p. 424)

9.366.4.4 receive_request() [1/2]

```
template<typename TReq , typename TRep >
bool  connext::Replier< TReq, TRep >::receive_request (
              connext::Sample< TReq > & request,
             const  DDS_Duration_t & max_wait )
```

Waits for a request and copies its contents into a **Sample** (p. 2183).

Equivalent to using **wait_for_requests(int, const Duration_t&)** (p. 2148) and **take_request(Sample<TReq>&)** (p. 2149)

See also

connext::Sample (p. 2183)

wait_for_requests(int, const Duration_t&) (p. 2148)

take_request(Sample<TReq>&) (p. 2149)

Basic Replier example (p. 425)

9.366.4.5 receive_request() [2/2]

```
template<typename TReq , typename TRep >
bool  connext::Replier< TReq, TRep >::receive_request (
              connext::SampleRef< TReq > request,
             const  DDS_Duration_t & max_wait )
```

Waits for a request and copies its contents into a **SampleRef** (p. 2191).

See also

connext::Requester::receive_reply(SampleRef<TRep>, const Duration_t&) (p. 2165)

9.366.4.6 receive_requests() [1/2]

```
template<typename TReq , typename TRep >
Replier< TReq, TRep >::LoanedSamplesType  connext::Replier< TReq, TRep >::receive_requests (
             const  DDS_Duration_t & max_wait )
```

Waits for multiple requests and provides a loaned container to access them.

Equivalent to using **receive_requests(int, int, const Duration_t&)** (p. 2147) with `min_count=1` and `max_count=DDS_LENGTH_UNLIMITED` (p. 636)

See also

connext::LoanedSamples (p. 1995)

receive_requests(int, int, const Duration_t&) (p. 2147)

9.366.4.7 receive_requests() [2/2]

```
template<typename TReq , typename TRep >
Replier< TReq, TRep >::LoanedSamplesType  connext::Replier< TReq, TRep >::receive_requests (
            int min_request_count,
            int max_request_count,
            const  DDS_Duration_t & max_wait )
```

Waits for multiple requests and provides a loaned container to access them.

Equivalent to using **wait_for_requests(int, const Duration_t&)** (p. 2148) and **take_requests(int)** (p. 2149)

See also

connext::LoanedSamples (p. 1995)

wait_for_requests(int, const Duration_t&) (p. 2148)

take_requests(int) (p. 2149)

9.366.4.8 wait_for_requests() [1/2]

```
template<typename TReq , typename TRep >
bool  connext::Replier< TReq, TRep >::wait_for_requests (
            const  DDS_Duration_t & max_wait )
```

Waits for requests.

Equivalent to **wait_for_requests(int, const Duration_t&)** (p. 2148) with `min_count=1`

See also

wait_for_requests(int, const Duration_t&) (p. 2148)

9.366.4.9 wait_for_requests() [2/2]

```
template<typename TReq , typename TRep >
bool  connext::Replier< TReq, TRep >::wait_for_requests (
            int min_count,
            const  DDS_Duration_t & max_wait )
```

Waits for requests.

This operation waits for min_count requests to be available. It will wait up to max_wait .

This operation is similar to **connext::Requester::wait_for_replies(int, const Duration_t&)** (p. 2167).

Parameters

min_count	Minimum number of requests that need to be available for this operation to unblock.
max_wait	Maximum waiting time after which this operation unblocks regardless of how many requests are available.

Returns

true if at least min_count requests were available before max_wait elapsed, or false otherwise.

See also

take_requests(int) (p. 2149)

connext::Requester::wait_for_replies(int, const Duration_t&) (p. 2167)

9.366.4.10 take_request() [1/2]

```
template<typename TReq , typename TRep >
bool  connext::Replier< TReq, TRep >::take_request (
              connext::Sample< TReq > & request )
```

Copies the contents of a request into a **Sample** (p. 2183).

See also

connext::Requester::take_reply(Sample<TRep>&) (p. 2169)

9.366.4.11 take_request() [2/2]

```
template<typename TReq , typename TRep >
bool  connext::Replier< TReq, TRep >::take_request (
              connext::SampleRef< TReq > request )
```

Copies the contents of a request into a **SampleRef** (p. 2191).

See also

connext::Requester::take_reply(SampleRef<TRep>) (p. 2169)

9.366.4.12 take_requests()

```
template<typename TReq , typename TRep >
Replier< TReq, TRep >::LoanedSamplesType  connext::Replier< TReq, TRep >::take_requests (
            int max_samples =  DDS_LENGTH_UNLIMITED )
```

Provides a loaned container to access the existing requests.

See also

connext::Requester::take_replies(int) (p. 2170)

9.366.4.13 read_request() [1/2]

```
template<typename TReq , typename TRep >
bool  connext::Replier< TReq, TRep >::read_request (
              connext::Sample< TReq > & request )
```

Copies the contents of a request into a **Sample** (p. 2183).

This operation is equivalent to **connext::Replier::take_request(Sample<TReq>&)** (p. 2149) except the request remains in the **Replier** (p. 2139) and can be read or taken again.

9.366.4.14 read_request() [2/2]

```
template<typename TReq , typename TRep >
bool  connext::Replier< TReq, TRep >::read_request (
              connext::SampleRef< TReq > request )
```

Copies the contents of a request into a **SampleRef** (p. 2191).

See also

connext::Requester::read_reply(SampleRef<TRep>) (p. 2175)

9.366.4.15 read_requests()

```
template<typename TReq , typename TRep >
Replier< TReq, TRep >::LoanedSamplesType  connext::Replier< TReq, TRep >::read_requests (
            int max_samples =  DDS_LENGTH_UNLIMITED )
```

Provides a loaned container to access the existing requests.

This operation is equivalent to **connext::Replier::take_requests(int)** (p. 2149) except the requests remain in the **Replier** (p. 2139) and can be read or taken again.

9.366.4.16 get_request_datareader()

```
template<typename TReq , typename TRep >
Replier< TReq, TRep > ::RequestDataReader * connext::Replier< TReq, TRep >::get_request_↩
datareader [inline]
```

Retrieves the underlying **DDSDataReader** (p. 1556).

See also

connext::Requester::get_reply_datareader() (p. 2177)

9.366.4.17 get_reply_datawriter()

```
template<typename TReq , typename TRep >
Replier< TReq, TRep > ::ReplyDataWriter * connext::Replier< TReq, TRep >::get_reply_datawriter
[inline]
```

Retrieves the underlying **DDSDataWriter** (p. 1589).

See also

connext::Requester::get_request_datawriter() (p. 2177)

9.367 connext::ReplierListener< TReq, TRep > Class Template Reference

Called when a **connext::Replier** (p. 2139) has new available requests.

Public Member Functions

- virtual void **on_request_available** (**Replier**< TReq, TRep > &replier)=0
  User callback.

9.367.1 Detailed Description

template<class TReq, class TRep>
class connext::ReplierListener< TReq, TRep >

Called when a **connext::Replier** (p. 2139) has new available requests.

A replier listener is a way to implement a callback that will be invoked when requests are available. It is an optional **connext::ReplierParams** (p. 2152).

This listener simply notifies when requests are available. The callback implementation can then use **connext::Replier↩ ::take_requests(int)** (p. 2149) to retrieve them.

A simpler callback mechanism, where one request sample is a parameter and the reply is the callback return value, is implemented by the **connext::SimpleReplier** (p. 2206).

See also

connext::Replier::Replier(const ReplierParams&) (p. 2144)

9.367.2 Member Function Documentation

9.367.2.1 on_request_available()

```
template<class TReq , class TRep >
virtual void  connext::ReplierListener< TReq, TRep >::on_request_available (
              Replier< TReq, TRep > & replier )  [pure virtual]
```

User callback.

See also

DDSDataReaderListener::on_data_available (p. 1585)

9.368 connext::ReplierParams< TReq, TRep > Class Template Reference

Contains the parameters for creating a **connext::Replier** (p. 2139).

Inherits connext::details::EntityParams.

Public Member Functions

- **ReplierParams** (**DDS::DomainParticipant** *participant)
  *Creates a **ReplierParams** (p. 2152) with the parameters that a **Replier** (p. 2139) always needs.*
- **ReplierParams** & **replier_listener** (**ReplierListener**< TReq, TRep > &listener)
  Sets a listener that is called when requests are available.
- **ReplierParams** & **service_name** (const std::string &service_name)
  *The service name the **Replier** (p. 2139) offers and Requesters use to match.*
- **ReplierParams** & **request_topic_name** (const std::string &req_topic)
  Sets a specific request topic name.
- **ReplierParams** & **reply_topic_name** (const std::string &rep_topic)
  Sets a specific reply topic name.
- **ReplierParams** & **qos_profile** (const std::string &qos_library_name, const std::string &qos_profile_name)
  Sets a QoS profile for the entities in this replier.
- **ReplierParams** & **datawriter_qos** (const **DDS_DataWriterQos** &qos)
  Sets the quality of service of the reply DataWriter.
- **ReplierParams** & **datareader_qos** (const **DDS_DataReaderQos** &qos)
  Sets the quality of service of the request DataReader.
- **ReplierParams** & **publisher** (**DDS::Publisher** *publisher)
  Sets a specific Publisher.
- **ReplierParams** & **subscriber** (**DDS::Subscriber** *subscriber)
  Sets a specific Subscriber.
- **ReplierParams** & **request_type_support** (**DDS::TypeSupport** *type_support)
  Sets the type support for the request type.
- **ReplierParams** & **reply_type_support** (**DDS::TypeSupport** *type_support)
  Sets the type support for the reply type.

9.368.1 Detailed Description

template<typename TReq, typename TRep>
class connext::ReplierParams< TReq, TRep >

Contains the parameters for creating a **connext::Replier** (p. 2139).

See also

connext::RequesterParams (p. 2178)

9.368.2 Constructor & Destructor Documentation

9.368.2.1 ReplierParams()

```
template<typename TReq , typename TRep >
connext::ReplierParams< TReq, TRep >::ReplierParams (
              DDS::DomainParticipant * participant )  [inline], [explicit]
```

Creates a **ReplierParams** (p. 2152) with the parameters that a **Replier** (p. 2139) always needs.

In addition to the participant , a **Replier** (p. 2139) needs either:

- A service name (**service_name** (p. 2154)), or
- Custom topic names (**reply_topic_name** (p. 2154) and **request_topic_name** (p. 2154)).

When **DDS_DynamicData** (p. 1013) is used, **connext::ReplierParams::request_type_support** (p. 2156) and/or **connext::ReplierParams::reply_type_support** (p. 2156) are required as well.

The other parameters are optional.

Parameters

participant	The DomainParticipant that this replier uses to join a domain.

9.368.3 Member Function Documentation

9.368.3.1 replier_listener()

```
template<typename TReq , typename TRep >
ReplierParams & connext::ReplierParams< TReq, TRep >::replier_listener (
             ReplierListener< TReq, TRep > & listener ) [inline]
```

Sets a listener that is called when requests are available.

See also

connext::ReplierListener (p. 2151)

9.368.3.2 service_name()

```
template<typename TReq , typename TRep >
ReplierParams & connext::ReplierParams< TReq, TRep >::service_name (
            const std::string & service_name ) [inline]
```

The service name the **Replier** (p. 2139) offers and Requesters use to match.

See also

connext::RequesterParams::service_name (p. 2180)

9.368.3.3 request_topic_name()

```
template<typename TReq , typename TRep >
ReplierParams & connext::ReplierParams< TReq, TRep >::request_topic_name (
            const std::string & req_topic ) [inline]
```

Sets a specific request topic name.

The specified topic name will be used, instead of allowing a topic name to be generated based on the **service_name** (p. 2154). If that topic already exists, it will be reused. This is useful to have the **Replier** (p. 2139) use a **DDSContent↩ FilteredTopic** (p. 1551) to receive only certain requests.

9.368.3.4 reply_topic_name()

```
template<typename TReq , typename TRep >
ReplierParams & connext::ReplierParams< TReq, TRep >::reply_topic_name (
            const std::string & rep_topic ) [inline]
```

Sets a specific reply topic name.

The specified topic name will be used, instead of allowing a topic name to be generated based on c the **service_name** (p. 2154). If that topic already exists, it will be reused.

9.368.3.5 qos_profile()

```
template<typename TReq , typename TRep >
ReplierParams &  connext::ReplierParams< TReq, TRep >::qos_profile (
            const std::string & qos_library_name,
            const std::string & qos_profile_name )  [inline]
```

Sets a QoS profile for the entities in this replier.

Specifies the XML QoS profile that will be used to configure the quality of service for the **Replier** (p. 2139)'s underlying reply DataWriter and request DataReader.

Parameters

qos_library_name	The name of the QoS library
qos_profile_name	The name of the QoS profile inside the QoS library

See also

connext::RequesterParams::qos_profile (p. 2180)

Configuring Request-Reply QoS profiles (p. 427)

Configuring QoS Profiles with XML (p. 390)

9.368.3.6 datawriter_qos()

```
template<typename TReq , typename TRep >
ReplierParams &  connext::ReplierParams< TReq, TRep >::datawriter_qos (
            const  DDS_DataWriterQos & qos )  [inline]
```

Sets the quality of service of the reply DataWriter.

See also

qos_profile (p. 2154)

9.368.3.7 datareader_qos()

```
template<typename TReq , typename TRep >
ReplierParams &  connext::ReplierParams< TReq, TRep >::datareader_qos (
            const  DDS_DataReaderQos & qos )  [inline]
```

Sets the quality of service of the request DataReader.

See also

qos_profile (p. 2154)

9.368.3.8 publisher()

```
template<typename TReq , typename TRep >
ReplierParams & connext::ReplierParams< TReq, TRep >::publisher (
            DDS::Publisher * publisher ) [inline]
```

Sets a specific Publisher.

See also

connext::RequesterParams::publisher (p. 2181)

9.368.3.9 subscriber()

```
template<typename TReq , typename TRep >
ReplierParams & connext::ReplierParams< TReq, TRep >::subscriber (
            DDS::Subscriber * subscriber ) [inline]
```

Sets a specific Subscriber.

See also

connext::RequesterParams::subscriber (p. 2182)

9.368.3.10 request_type_support()

```
template<typename TReq , typename TRep >
ReplierParams & connext::ReplierParams< TReq, TRep >::request_type_support (
            DDS::TypeSupport * type_support ) [inline]
```

Sets the type support for the request type.

This parameter is required when TReq is **DDS_DynamicData** (p. 1013).

In all other cases, this parameter is ignored and the singleton Reply<TReq, TRep>::RequestTypeSupport is used instead

See also

DDSDynamicDataTypeSupport (p. 1725)

9.368.3.11 reply_type_support()

```
template<typename TReq , typename TRep >
ReplierParams & connext::ReplierParams< TReq, TRep >::reply_type_support (
            DDS::TypeSupport * type_support ) [inline]
```

Sets the type support for the reply type.

This parameter is required when TRep is **DDS_DynamicData** (p. 1013).

In all other cases, this parameter is ignored and the singleton Reply<TReq, TRep>::RequestTypeSupport is used instead

See also

DDSDynamicDataTypeSupport (p. 1725)

9.369 connext::Requester< TReq, TRep > Class Template Reference

Allows sending requests and receiving replies.

Public Types

- typedef TReq **Request**

  The request type.
- typedef TRep **Reply**

  The reply type.
- typedef dds_type_traits< TReq > **::DataWriter RequestDataWriter**

  The typed DataWriter for type TReq.
- typedef dds_type_traits< TRep > **::DataReader ReplyDataReader**

  The typed DataReader for type TRep.
- typedef dds_type_traits< TReq > **::TypeSupport RequestTypeSupport**

  The type support for type TReq.
- typedef dds_type_traits< TRep > **::TypeSupport ReplyTypeSupport**

  The type support for type TRep.

Public Member Functions

- **Requester** (**DDSDomainParticipant** *participant, const std::string &service_name)

  *Creates a **Requester** (p. 2157) with the minimum set of parameters.*
- **Requester** (const **RequesterParams** &params)

  *Creates a **Requester** (p. 2157) with parameters.*
- virtual ~**Requester** ()

  Releases the internal entities created by this object.
- template<typename UReq >
  void **send_request** (const UReq &request)

  Sends a request.
- template<typename UReq >
  void **send_request** (**WriteSample**< UReq > &request)

  Sends a request and gets back information about it that allows correlation with future replies.
- template<typename UReq >
  void **send_request** (**WriteSampleRef**< UReq > &request)

  Sends a request and gets back information about it that allows correlation with future replies.
- bool **receive_reply** (**Sample**< TRep > &reply, const **DDS_Duration_t** &timeout)

  *Waits for a reply and copies its contents into a **Sample** (p. 2183).*
- bool **receive_reply** (**SampleRef**< TRep > reply, const **DDS_Duration_t** &timeout)

  *Waits for a reply and copies its contents into a **SampleRef** (p. 2191).*
- **LoanedSamples**< TRep > **receive_replies** (const **DDS_Duration_t** &max_wait)

  Waits for multiple replies and provides a loaned container to access them.
- **LoanedSamples**< TRep > **receive_replies** (int min_count, int max_count, const **DDS_Duration_t** &max_↩ wait)

  Waits for multiple replies and provides a loaned container to access them.
- bool **wait_for_replies** (int min_count, const **DDS_Duration_t** &max_wait)

  Waits for replies to any request.
- bool **wait_for_replies** (const **DDS_Duration_t** &max_wait)

  Waits for replies to any request.
- bool **wait_for_replies** (int min_count, const **DDS_Duration_t** &max_wait, const **SampleIdentity_t** &related↩ _request_id)

  Waits for replies to a specific request.
- bool **take_reply** (**Sample**< TRep > &reply)

  *Copies the contents of a reply into a **Sample** (p. 2183).*
- bool **take_reply** (**SampleRef**< TRep > reply)

  *Copies the contents of a reply into a **SampleRef** (p. 2191).*
- **LoanedSamples**< TRep > **take_replies** (int max_count= **DDS_LENGTH_UNLIMITED**)

  Provides a loaned container to access the existing replies.
- bool **take_reply** (**Sample**< TRep > &reply, const **SampleIdentity_t** &related_request_id)

  Copies the contents of a reply for a specific request.
- bool **take_reply** (**SampleRef**< TRep > reply, const **SampleIdentity_t** &related_request_id)

  Copies the contents of a reply for a specific request.
- **LoanedSamples**< TRep > **take_replies** (int max_count, const **SampleIdentity_t** &related_request_id)

  Provides a loaned container to access the existing replies for a specific request.
- **LoanedSamples**< TRep > **take_replies** (const **SampleIdentity_t** &related_request_id)

  Provides a loaned container to access the existing replies for a specific request.
- bool **read_reply** (**Sample**< TRep > &reply)

*Copies the contents of a reply into a **Sample** (p. 2183).*

- bool **read_reply** (**SampleRef**< TRep > reply)

  *Copies the contents of a reply into a **SampleRef** (p. 2191).*

- **LoanedSamples**< TRep > **read_replies** (int max_count= **DDS_LENGTH_UNLIMITED**)

  Provides a loaned container to access the existing replies.

- bool **read_reply** (**Sample**< TRep > &reply, const **SampleIdentity_t** &related_request_id)

  Copies the contents of a reply for a specific request.

- bool **read_reply** (**SampleRef**< TRep > reply, const **SampleIdentity_t** &related_request_id)

  Copies the contents of a reply for a specific request.

- **LoanedSamples**< TRep > **read_replies** (int max_count, const **SampleIdentity_t** &related_request_id)

  Provides a loaned container to access the existing replies for a specific request.

- **LoanedSamples**< TRep > **read_replies** (const **SampleIdentity_t** &related_request_id)

  Provides a loaned container to access the existing replies for a specific request.

- **RequestDataWriter** * **get_request_datawriter** ()

  *Retrieves the underlying **DDSDataWriter** (p. 1589).*

- **ReplyDataReader** * **get_reply_datareader** ()

  *Retrieves the underlying **DDSDataReader** (p. 1556).*

9.369.1 Detailed Description

template<typename TReq, typename TRep>
class connext::Requester< TReq, TRep >

Allows sending requests and receiving replies.

A requester is an entity with two associated **topics** (p. 63): a request topic and a reply topic. It can send requests by publishing samples of the request topic and receive replies to those requests by subscribing to the reply topic.

Valid types for these topics (`TReq` and `TRep`) are: those generated by rtiddsgen, the **DDS built-in types** (p. 95), and **DDS_DynamicData** (p. 1013).

For example:

```
using namespace connext;


Requester<Foo, Bar> requester(...);

Requester<DDS::DynamicData, Bar> requester(...);

Requester<Foo, DDS::Octets> requester(...);
```

A **Replier** (p. 2139) and a **Requester** (p. 2157) communicate when they use the same topics for requests and replies (see **connext::RequesterParams::service_name** (p. 2180)) on the same **domain** (p. 53).

A **Requester** (p. 2157) can send requests and receive one or multiple replies. It does that using the following operations:

- Sending requests (i.e. publishing request samples on the request topic)
- Waiting for replies to be received by the middleware (for any request or for a specific request)

- Getting those replies from the middleware. There are two ways to do this: take (the data samples are removed from the middleware), read (the data samples remain in the middleware and can be read or taken again).
- A convenience operation, receive (which is a combination of wait and take).

In all cases, the middleware guarantees that a requester only receives reply samples that are associated with those requests that it sends.

For multi-reply scenarios, in which a **Requester** (p. 2157) receives multiple replies from a **Replier** (p. 2139) for a given request, the **Requester** (p. 2157) can check if a reply is the last reply of a sequence of replies. To do so, see if the bit **DDS_INTERMEDIATE_REPLY_SEQUENCE_SAMPLE** (p. 673) is set in **DDS_SampleInfo::flag** (p. 1353) after receiving each reply. This indicates it is NOT the last reply.

A requester has an associated **DDSDomainParticipant** (p. 1619), which can be shared with other requesters or RTI Connext routines. All the other RTI Connext entities required for the request-reply interaction, including a **DDSData↩ Writer** (p. 1589) for writing requests and a **DDSDataReader** (p. 1556) for reading replies, are automatically created when the requester is constructed.

Quality of Service for the underlying DataWriter and DataReader can be configured (see **connext::RequesterParams↩ ::qos_profile** (p. 2180)). By default, they are created with **DDS_RELIABLE_RELIABILITY_QOS** (p. 633). The exact default configuration is described here: **Configuring Request-Reply QoS profiles** (p. 427)

Template Parameters

TReq	The data type for the request topic
TRep	The data type for the reply topic

See also

connext::Replier (p. 2139)

Request-Reply Examples (p. 419)

Basic Requester example (p. 421)

9.369.2 Member Typedef Documentation

9.369.2.1 Request

```
template<typename TReq , typename TRep >
typedef TReq  connext::Requester< TReq, TRep >::Request
```

The request type.

9.369.2.2 Reply

```
template<typename TReq , typename TRep >
typedef TRep  connext::Requester< TReq, TRep >::Reply
```

The reply type.

9.369.2.3 RequestDataWriter

```
template<typename TReq , typename TRep >
typedef dds_type_traits<TReq> ::DataWriter  connext::Requester< TReq, TRep >::RequestDataWriter
```

The typed DataWriter for type `TReq`.

9.369.2.4 ReplyDataReader

```
template<typename TReq , typename TRep >
typedef dds_type_traits<TRep> ::DataReader  connext::Requester< TReq, TRep >::ReplyDataReader
```

The typed DataReader for type `TRep`.

9.369.2.5 RequestTypeSupport

```
template<typename TReq , typename TRep >
typedef dds_type_traits<TReq> ::TypeSupport  connext::Requester< TReq, TRep >::RequestType↩
Support
```

The type support for type `TReq`.

9.369.2.6 ReplyTypeSupport

```
template<typename TReq , typename TRep >
typedef dds_type_traits<TRep> ::TypeSupport  connext::Requester< TReq, TRep >::ReplyTypeSupport
```

The type support for type `TRep`.

9.369.3 Constructor & Destructor Documentation

9.369.3.1 Requester() [1/2]

```
template<typename TReq , typename TRep >
connext::Requester< TReq, TRep >::Requester (
              DDSDomainParticipant * participant,
             const std::string & service_name )
```

Creates a **Requester** (p. 2157) with the minimum set of parameters.

Parameters

participant	The DomainParticipant this requester uses to join a DDS domain
service_name	The service name. See **connext::RequesterParams::service_name** (p. 2180)

Exceptions

One	of the **RTI Connext Exceptions** (p. 39)

9.369.3.2 Requester() [2/2]

```
template<typename TReq , typename TRep >
connext::Requester< TReq, TRep >::Requester (
            const  RequesterParams & params )  [explicit]
```

Creates a **Requester** (p. 2157) with parameters.

Parameters

params	All the parameters that configure this requester. See **connext::RequesterParams** (p. 2178) for the list of mandatory parameters.

Exceptions

One	of the **RTI Connext Exceptions** (p. 39)

See also

connext::RequesterParams (p. 2178)

Requester Creation (p. 421)

9.369.3.3 ~Requester()

```
template<typename TReq , typename TRep >
connext::Requester< TReq, TRep >::~ Requester  [virtual]
```

Releases the internal entities created by this object.

Among other internal resources, it deletes the underlying DataReader and DataWriter.

See also

DDSSubscriber::delete_datareader (p. 1870)

DDSPublisher::delete_datawriter (p. 1829)

9.369.4 Member Function Documentation

9.369.4.1 send_request() [1/3]

```
template<typename TReq , typename TRep >
template<typename UReq >
void  connext::Requester< TReq, TRep >::send_request (
            const UReq & request )
```

Sends a request.

If a future reply needs to be correlated to exactly this request, use **connext::Requester::send_request(WriteSample<UReq>&)** (p. 2163).

Parameters

request	The request to be sent

Exceptions

One	of the **RTI Connext Exceptions** (p. 39)

MT Safety:

SAFE

See also

send_request(WriteSample<UReq>&) (p. 2163)

9.369.4.2 send_request() [2/3]

```
template<typename TReq , typename TRep >
template<typename UReq >
void  connext::Requester< TReq, TRep >::send_request (
            WriteSample< UReq > & request )
```

Sends a request and gets back information about it that allows correlation with future replies.

After calling this operation, the sample contains a valid identity that can be used for correlation with future replies.

See example code in **Correlating requests and replies** (p. 423).

Parameters

request	<<***inout***>> *(*p. *433)* Contains the request and optional write parameters. When this call ends successfully, **connext::WriteSample** (p. 2219) contains a valid identity that can be used for correlation with future replies.

Exceptions

One	of the **RTI Connext Exceptions** (p. 39) ; **connext::TimeoutException** (p. 2217) may be reported in the same conditions as in **FooDataWriter::write** (p. 1951).

MT Safety:

SAFE

See also

connext::WriteSample::identity() (p. 2222)

wait_for_replies(int, const Duration_t&, const SampleIdentity_t&) (p. 2168)

take_replies(int, const SampleIdentity_t&) (p. 2172)

Correlating requests and replies (p. 423)

9.369.4.3 send_request() [3/3]

```
template<typename TReq , typename TRep >
template<typename UReq >
void  connext::Requester< TReq, TRep >::send_request (
             WriteSampleRef< UReq > & request )
```

Sends a request and gets back information about it that allows correlation with future replies.

This operation is equivalent to **connext::Requester::send_request(WriteSample<UReq>&)** (p. 2163) except it uses a **WriteSampleRef** (p. 2223).

A **WriteSampleRef** (p. 2223) needs to be initialized with references to existing data and info objects before it can be used. See an example here: **Basic Requester example using SampleRef** (p. 424)

See also

connext::Requester::send_request(WriteSample<UReq>&) (p. 2163)

Basic Requester example using SampleRef (p. 424)

9.369.4.4 receive_reply() [1/2]

```
template<typename TReq , typename TRep >
bool  connext::Requester< TReq, TRep >::receive_reply (
              Sample< TRep > & reply,
             const  DDS_Duration_t & timeout )  [inline]
```

Waits for a reply and copies its contents into a **Sample** (p. 2183).

This operation is equivalent to using **wait_for_replies(int, const Duration_t&)** (p. 2167) and **take_reply(Sample<TRep>&)** (p. 2169).

MT Safety:

Same restrictions as **wait_for_replies(int, const Duration_t&)** (p. 2167)

See also

connext::Sample (p. 2183)

wait_for_replies(int, const Duration_t&) (p. 2167)

take_reply(Sample<TRep>&) (p. 2169)

Basic Requester example (p. 421)

9.369.4.5 receive_reply() [2/2]

```
template<typename TReq , typename TRep >
bool  connext::Requester< TReq, TRep >::receive_reply (
              SampleRef< TRep > reply,
             const  DDS_Duration_t & timeout )  [inline]
```

Waits for a reply and copies its contents into a **SampleRef** (p. 2191).

This operation is equivalent to using **wait_for_replies(int, const Duration_t&)** (p. 2167) and **take_reply(SampleRef<TRep>)** (p. 2169).

MT Safety:

Same restrictions as **wait_for_replies(int, const Duration_t&)** (p. 2167)

See also

connext::SampleRef (p. 2191)

wait_for_replies(int, const Duration_t&) (p. 2167)

take_reply(SampleRef<TRep>) (p. 2169)

Basic Requester example using SampleRef (p. 424)

9.369.4.6 receive_replies() [1/2]

```
template<typename TReq , typename TRep >
LoanedSamples< TRep >  connext::Requester< TReq, TRep >::receive_replies (
            const  DDS_Duration_t & max_wait )
```

Waits for multiple replies and provides a loaned container to access them.

This operation is equivalent to using **receive_replies(int,int,const Duration_t&)** (p. 2166) with `min_count=1` and `max_count=`**DDS_LENGTH_UNLIMITED** (p. 636)

MT Safety:

See **wait_for_replies(int, const Duration_t&)** (p. 2167)

See also

connext::LoanedSamples (p. 1995)

receive_replies(int,int,const Duration_t&) (p. 2166)

9.369.4.7 receive_replies() [2/2]

```
template<typename TReq , typename TRep >
LoanedSamples< TRep >  connext::Requester< TReq, TRep >::receive_replies (
            int min_count,
            int max_count,
            const  DDS_Duration_t & max_wait )
```

Waits for multiple replies and provides a loaned container to access them.

This operation is equivalent to using **wait_for_replies(int, const Duration_t&)** (p. 2167) and **take_replies(int)** (p. 2170).

MT Safety:

See **wait_for_replies(int, const Duration_t&)** (p. 2167)

Exceptions

One	of the **RTI Connext Exceptions** (p. 39)

See also

connext::LoanedSamples (p. 1995)

wait_for_replies(int, const Duration_t&) (p. 2167)

take_replies(int) (p. 2170)

9.369.4.8 wait_for_replies() [1/3]

```
template<typename TReq , typename TRep >
bool  connext::Requester< TReq, TRep >::wait_for_replies (
            int min_count,
            const  DDS_Duration_t & max_wait )
```

Waits for replies to any request.

This operation waits for min_count requests to be available for up to max_wait .

If this operation is called several times but the available replies are not taken (with **take_replies(int)** (p. 2170)), this operation may return immediately and will not wait for new replies. New replies may replace existing ones if they are not taken, depending on the **DDS_HistoryQosPolicy** (p. 1162) used to configure this **Requester** (p. 2157).

Parameters

min_count	Minimum number of replies that need to be available for this operation to unblock.
max_wait	Maximum waiting time after which this operation unblocks, regardless of how many replies are available.

Returns

true if at least min_count replies were available before max_wait elapsed, or false otherwise.

MT Safety:

Concurrent calls to this operation on the same object are not allowed. However, waiting for replies for specific requests in parallel is supported (see **wait_for_replies(int, const Duration_t&, const SampleIdentity_t&)** (p. 2168)).

See also

take_replies(int) (p. 2170)

9.369.4.9 wait_for_replies() [2/3]

```
template<typename TReq , typename TRep >
bool  connext::Requester< TReq, TRep >::wait_for_replies (
            const  DDS_Duration_t & max_wait )
```

Waits for replies to any request.

This operation is equivalent to using **wait_for_replies(int, const Duration_t&)** (p. 2167) with `min_count=1`.

See also

wait_for_replies(int, const Duration_t&) (p. 2167)

9.369.4.10 wait_for_replies() [3/3]

```
template<typename TReq , typename TRep >
bool  connext::Requester< TReq, TRep >::wait_for_replies (
            int min_count,
            const  DDS_Duration_t & max_wait,
            const  SampleIdentity_t & related_request_id )
```

Waits for replies to a specific request.

This operation is analogous to **wait_for_replies(int, const Duration_t&)** (p. 2167) except this operation waits for replies for a specific request.

Parameters

min_count	Minimum number of replies for the related request that need to be available for this operation to unblock.
max_wait	Maximum wait time after which this operation unblocks, regardless of how many replies are available.
related_request_id	The identity of a request previously sent by this **Requester** (p. 2157)

Returns

true if at least min_count replies for the related request were available before max_wait elapsed, or false otherwise.

Exceptions

One	of the **RTI Connext Exceptions** (p. 39)

MT Safety:

SAFE

See also

wait_for_replies(int, const Duration_t&) (p. 2167)

9.369.4.11 take_reply() [1/4]

```
template<typename TReq , typename TRep >
bool  connext::Requester< TReq, TRep >::take_reply (
              Sample< TRep > & reply )  [inline]
```

Copies the contents of a reply into a **Sample** (p. 2183).

Takes a reply sample from the **Requester** (p. 2157) and makes a copy.

This operation may be used after a call to **wait_for_replies(int, const Duration_t&)** (p. 2167).

To avoid copies, you can use **take_replies(int)** (p. 2170).

Parameters

reply	The sample where the reply data and the related SampleInfo are copied

Returns

true if there was a reply to get. If this operation returns false, the contents of `reply` remain unchanged, except **DDS_SampleInfo::valid_data** (p. 1351) becomes false.

Exceptions

One	of the **RTI Connext Exceptions** (p. 39)

MT Safety:

SAFE

See also

connext::Sample (p. 2183)

wait_for_replies(int, const Duration_t&) (p. 2167)

9.369.4.12 take_reply() [2/4]

```
template<typename TReq , typename TRep >
bool  connext::Requester< TReq, TRep >::take_reply (
              SampleRef< TRep > reply )  [inline]
```

Copies the contents of a reply into a **SampleRef** (p. 2191).

This operation is analogous to **connext::Requester::take_reply(Sample<TRep>&)** (p. 2169) except it uses a **SampleRef** (p. 2191) instance.

A **SampleRef** (p. 2191) needs to be initialized with references to existing data and info objects before it can be used. See an example here: **Basic Requester example using SampleRef** (p. 424)

MT Safety:

SAFE

See also

connext::SampleRef (p. 2191)

take_reply(Sample<TRep>&) (p. 2169)

Basic Requester example using SampleRef (p. 424)

9.369.4.13 take_replies() [1/3]

```
template<typename TReq , typename TRep >
LoanedSamples< TRep >  connext::Requester< TReq, TRep >::take_replies (
             int max_count =  DDS_LENGTH_UNLIMITED )
```

Provides a loaned container to access the existing replies.

Takes all the existing replies up to `max_count` and provides a loaned container to access them.

This operation does not make a copy of the data.

The loan is returned with **connext::LoanedSamples::return_loan** (p. 2002) or in the destructor

This operation may be used after a call to **wait_for_replies(int, const Duration_t&)** (p. 2167)

See an **example** here: **Taking loaned samples** (p. 422)

Parameters

max_count	The maximum number of samples that are taken at a time. The special value **DDS_LENGTH_UNLIMITED** (p. 636) may be used. This value will read up to the limit specified by **DDS_DataReaderResourceLimitsQosPolicy::max_samples_per_read** (p. 873)

Returns

A container with up to max_count elements. May be empty if there were no replies to get.

Exceptions

One	of the **RTI Connext Exceptions** (p. 39)

MT Safety:

SAFE

See also

connext::Sample (p. 2183)

connext::LoanedSamples (p. 1995)

connext::LoanedSamples::return_loan (p. 2002)

wait_for_replies(int, const Duration_t&) (p. 2167)

FooDataReader::take (p. 1920) (for a more detailed description on how QoS and other parameters affect the underlying DataReader)

Taking loaned samples (p. 422)

Taking samples by copy (p. 422)

9.369.4.14 take_reply() [3/4]

```
template<typename TReq , typename TRep >
bool  connext::Requester< TReq, TRep >::take_reply (
             Sample< TRep > & reply,
            const  SampleIdentity_t & related_request_id )  [inline]
```

Copies the contents of a reply for a specific request.

This operation is analogous to **take_reply(Sample<TRep>&)** (p. 2169) except the reply corresponds to a specific request.

Parameters

reply	The sample where a reply is copied into
related_request_id	The identity of a request previously sent by this **Requester** (p. 2157)

Returns

true if there was a reply to get. If this operation returns false, the contents of `reply` remain unchanged, except **DDS_SampleInfo::valid_data** (p. 1351) becomes false.

MT Safety:

SAFE

See also

connext::Sample (p. 2183)

take_reply(Sample<TRep>&) (p. 2169)

send_request(WriteSample<UReq>&) (p. 2163)

Correlating requests and replies (p. 423)

9.369.4.15 take_reply() [4/4]

```
template<typename TReq , typename TRep >
bool  connext::Requester< TReq, TRep >::take_reply (
              SampleRef< TRep > reply,
             const  SampleIdentity_t & related_request_id )  [inline]
```

Copies the contents of a reply for a specific request.

This operation is analogous to **connext::Requester::take_reply(Sample<TRep>&, const SampleIdentity_t&)** (p. 2171) except it uses a **SampleRef** (p. 2191) instance.

A **SampleRef** (p. 2191) needs to be initialized with references to existing data and info objects before it can be used. See an example here: **Basic Requester example using SampleRef** (p. 424)

MT Safety:

SAFE

See also

connext::SampleRef (p. 2191)

take_reply(SampleRef<TRep>) (p. 2169)

take_reply(Sample<TRep>&, const SampleIdentity_t&) (p. 2171)

Basic Requester example using SampleRef (p. 424)

Correlating requests and replies (p. 423)

9.369.4.16 take_replies() [2/3]

```
template<typename TReq , typename TRep >
LoanedSamples< TRep >  connext::Requester< TReq, TRep >::take_replies (
            int max_count,
            const  SampleIdentity_t & related_request_id )
```

Provides a loaned container to access the existing replies for a specific request.

This operation is analogous to **take_replies(int)** (p. 2170) except the replies this operation provides correspond to a specific request.

Parameters

max_count	The maximum number of samples that are taken at a time. The special value **DDS_LENGTH_UNLIMITED** (p. 636) may be used.
related_request_id	The identity of a request previously sent by this **Requester** (p. 2157)

MT Safety:

SAFE

See also

take_replies(int) (p. 2170)

connext::LoanedSamples (p. 1995)

Taking samples by copy (p. 422)

Correlating requests and replies (p. 423)

9.369.4.17 take_replies() [3/3]

```
template<typename TReq , typename TRep >
LoanedSamples< TRep >  connext::Requester< TReq, TRep >::take_replies (
            const  SampleIdentity_t & related_request_id )
```

Provides a loaned container to access the existing replies for a specific request.

Equivalent to using **take_replies(int, const SampleIdentity_t&)** (p. 2172) with `max_count=`**DDS_LENGTH_UNLIMITED** (p. 636)

MT Safety:

SAFE

See also

connext::LoanedSamples (p. 1995)

take_replies(int, const SampleIdentity_t&) (p. 2172)

References **DDS_LENGTH_UNLIMITED**.

9.369.4.18 read_reply() [1/4]

```
template<typename TReq , typename TRep >
bool  connext::Requester< TReq, TRep >::read_reply (
              Sample< TRep > & reply )  [inline]
```

Copies the contents of a reply into a **Sample** (p. 2183).

This operation is equivalent to **connext::Requester::take_reply(Sample<TRep>&)** (p. 2169) except the reply remains in the **Requester** (p. 2157) and can be read or taken again.

9.369.4.19 read_reply() [2/4]

```
template<typename TReq , typename TRep >
bool  connext::Requester< TReq, TRep >::read_reply (
              SampleRef< TRep > reply )  [inline]
```

Copies the contents of a reply into a **SampleRef** (p. 2191).

This operation is analogous to **connext::Requester::read_reply(Sample<TRep>&)** (p. 2174) except it uses a **SampleRef** (p. 2191) instance.

A **SampleRef** (p. 2191) needs to be initialized with references to existing data and info objects before it can be used. See an example here: **Basic Requester example using SampleRef** (p. 424)

MT Safety:

SAFE

See also

connext::SampleRef (p. 2191)

read_reply(Sample<TRep>&) (p. 2174)

Basic Requester example using SampleRef (p. 424)

9.369.4.20 read_replies() [1/3]

```
template<typename TReq , typename TRep >
LoanedSamples< TRep >  connext::Requester< TReq, TRep >::read_replies (
             int max_count =  DDS_LENGTH_UNLIMITED )
```

Provides a loaned container to access the existing replies.

This operation is equivalent to **connext::Requester::take_replies(int)** (p. 2170) except the replies remain in the **Requester** (p. 2157) and can be read or taken again.

9.369.4.21 read_reply() [3/4]

```
template<typename TReq , typename TRep >
bool  connext::Requester< TReq, TRep >::read_reply (
             Sample< TRep > & reply,
            const  SampleIdentity_t & related_request_id )  [inline]
```

Copies the contents of a reply for a specific request.

This operation is equivalent to **connext::Requester::take_reply(Sample<TRep>&, const SampleIdentity_t&)** (p. 2171) except the reply remains in the **Requester** (p. 2157) and can be read or taken again.

9.369.4.22 read_reply() [4/4]

```
template<typename TReq , typename TRep >
bool  connext::Requester< TReq, TRep >::read_reply (
             SampleRef< TRep > reply,
            const  SampleIdentity_t & related_request_id )  [inline]
```

Copies the contents of a reply for a specific request.

This operation is analogous to **connext::Requester::read_reply(Sample<TRep>&, const SampleIdentity_t&)** (p. 2175) except it uses a **SampleRef** (p. 2191) instance.

A **SampleRef** (p. 2191) needs to be initialized with references to existing data and info objects before it can be used. See an example here: **Basic Requester example using SampleRef** (p. 424)

MT Safety:

SAFE

See also

connext::SampleRef (p. 2191)

read_reply(SampleRef<TRep>) (p. 2175)

read_reply(Sample<TRep>&, const SampleIdentity_t&) (p. 2175)

Basic Requester example using SampleRef (p. 424)

Correlating requests and replies (p. 423)

9.369.4.23 read_replies() [2/3]

```
template<typename TReq , typename TRep >
LoanedSamples< TRep >  connext::Requester< TReq, TRep >::read_replies (
            int max_count,
            const  SampleIdentity_t & related_request_id )
```

Provides a loaned container to access the existing replies for a specific request.

This operation is equivalent to **connext::Requester::take_replies(int, const SampleIdentity_t&)** (p. 2172) except the replies remain in the **Requester** (p. 2157) and can be read or taken again.

9.369.4.24 read_replies() [3/3]

```
template<typename TReq , typename TRep >
LoanedSamples< TRep >  connext::Requester< TReq, TRep >::read_replies (
            const  SampleIdentity_t & related_request_id )
```

Provides a loaned container to access the existing replies for a specific request.

This operation is equivalent to **connext::Requester::take_replies(const SampleIdentity_t&)** (p. 2174) except the replies remain in the **Requester** (p. 2157) and can be read or taken again.

References **DDS_LENGTH_UNLIMITED**.

9.369.4.25 get_request_datawriter()

```
template<typename TReq , typename TRep >
Requester< TReq, TRep > ::RequestDataWriter *  connext::Requester< TReq, TRep >::get_request_↩
datawriter  [inline]
```

Retrieves the underlying **DDSDataWriter** (p. 1589).

Accessing the request DataWriter may be useful for a number of advanced use cases, such as:

- Finding matching subscriptions (e.g., Repliers)
- Setting a DataWriter listener
- Getting DataWriter protocol or cache statuses

MT Safety:

SAFE

See also

DDSDataWriter (p. 1589)
FooDataWriter (p. 1944)
DDSDataWriter::get_matched_subscriptions (p. 1599)
DDSDataWriter::get_matched_subscription_data (p. 1600)
DDSDataWriter::set_listener (p. 1607)
DDSDataWriter::get_datawriter_protocol_status (p. 1595)

9.369.4.26 get_reply_datareader()

```
template<typename TReq , typename TRep >
Requester< TReq, TRep > ::ReplyDataReader * connext::Requester< TReq, TRep >::get_reply_↩
datareader [inline]
```

Retrieves the underlying **DDSDataReader** (p. 1556).

Accessing the reply DataReader may be useful for a number of advanced use cases, such as:

- Finding matching publications (e.g., Repliers)
- Setting a DataReader listener
- Getting DataReader protocol or cache statuses

MT Safety:

SAFE

See also

DDSDataReader (p. 1556)
FooDataReader (p. 1917)
DDSDataReader::get_matched_publications (p. 1567)
DDSDataReader::get_matched_publication_data (p. 1568)
DDSDataReader::set_listener (p. 1577)
DDSDataReader::get_datareader_protocol_status (p. 1573)

9.370 connext::RequesterParams Class Reference

Contains the parameters for creating a **connext::Requester** (p. 2157).

Inherits connext::details::EntityParams.

Public Member Functions

- **RequesterParams** (**DomainParticipant** *participant)
  *Creates a **RequesterParams** (p. 2178) with the parameters a **Requester** (p. 2157) always needs.*
- **RequesterParams** & **service_name** (const std::string &name)
  *The service name that Repliers and **Requester** (p. 2157) use to match and communicate.*
- **RequesterParams** & **request_topic_name** (const std::string &name)
  Sets a specific request topic name.
- **RequesterParams** & **reply_topic_name** (const std::string &name)
  Sets a specific reply topic name.
- **RequesterParams** & **qos_profile** (const std::string &qos_library_name, const std::string &qos_profile_name)

Sets a QoS profile for the entities in this requester.

- **RequesterParams** & **datawriter_qos** (const **DDS_DataWriterQos** &qos)

  Sets the quality of service of the request DataWriter.
- **RequesterParams** & **datareader_qos** (const **DDS_DataReaderQos** &qos)

  Sets the quality of service of the request DataReader.
- **RequesterParams** & **publisher** (**DDSPublisher** ∗publisher)

  Sets a specific Publisher.
- **RequesterParams** & **subscriber** (**DDSSubscriber** ∗subscriber)

  Sets a specific Subscriber.
- **RequesterParams** & **request_type_support** (**TypeSupport** ∗type_support)

  Sets the type support for the request type.
- **RequesterParams** & **reply_type_support** (**TypeSupport** ∗type_support)

  Sets the type support for the reply type.

9.370.1 Detailed Description

Contains the parameters for creating a **connext::Requester** (p. 2157).

See also

Creating a Requester with optional parameters (p. 421)

9.370.2 Constructor & Destructor Documentation

9.370.2.1 RequesterParams()

```
connext::RequesterParams::RequesterParams (
              DomainParticipant * participant )  [explicit]
```

Creates a **RequesterParams** (p. 2178) with the parameters a **Requester** (p. 2157) always needs.

In addition to the participant , a **Requester** (p. 2157) needs either:

- A service name (**service_name** (p. 2180)),
- Or custom topic names (**request_topic_name** (p. 2180) and **reply_topic_name** (p. 2180))

When **DDS_DynamicData** (p. 1013) is used, **connext::RequesterParams::request_type_support** (p. 2182) and/or **connext::RequesterParams::reply_type_support** (p. 2182) are required as well.

The rest of the parameters that can be set in a **RequesterParams** (p. 2178) object are optional.

Parameters

participant	The **DDSDomainParticipant** (p. 1619) this requester uses to join a domain.

See also

Creating a Requester with optional parameters (p. 421)

9.370.3 Member Function Documentation

9.370.3.1 service_name()

```
RequesterParams & connext::RequesterParams::service_name (
            const std::string & name )
```

The service name that Repliers and **Requester** (p. 2157) use to match and communicate.

A **Requester** (p. 2157) and a **Replier** (p. 2139) need to be configured with the same topic names in order to match.

The service name is used to generate a request topic and a reply topic that the **Requester** (p. 2157) and **Replier** (p. 2139) will use to communicate. For example, the service name "MyService" will be used to create topics named "MyServiceRequest" and "MyServiceReply".

In some cases, the name of these topics is known beforehand or needs to be customized for another reason. The service name can be overridden by setting specific request and reply topic names using **request_topic_name** (p. 2180) and **reply_topic_name** (p. 2180).

9.370.3.2 request_topic_name()

```
RequesterParams & connext::RequesterParams::request_topic_name (
            const std::string & name )
```

Sets a specific request topic name.

The specified topic name will be used, instead of allowing a topic name to be generated based on the **service_name** (p. 2180). If that topic already exists, it will be reused.

9.370.3.3 reply_topic_name()

```
RequesterParams & connext::RequesterParams::reply_topic_name (
            const std::string & name )
```

Sets a specific reply topic name.

The specified topic name will be used, instead of allowing a topic name to be generated based on the **service_name** (p. 2180). If that topic already exists, it will be reused.

9.370.3.4 qos_profile()

```
RequesterParams & connext::RequesterParams::qos_profile (
            const std::string & qos_library_name,
            const std::string & qos_profile_name )
```

Sets a QoS profile for the entities in this requester.

Specifies an XML QoS profile that will be used to configure the quality of service of a **Requester** (p. 2157)'s underlying request DataWriter and reply DataReader.

Within that profile, the DataWriter QoS in <datawriter_qos> will be used to configure the request DataWriter and the DataReader.

Alternatively, you can set the QoS using the DataWriterQos and DataReaderQos objects (**connext::Requester↩ Params::datawriter_qos** (p. 2181), **connext::RequesterParams::datareader_qos** (p. 2181)).

Parameters

qos_library_name	The name of the QoS library
qos_profile_name	The name of the QoS profile inside the QoS library

See also

Configuring Request-Reply QoS profiles (p. 427)

Configuring QoS Profiles with XML (p. 390)

9.370.3.5 datawriter_qos()

```
RequesterParams & connext::RequesterParams::datawriter_qos (
            const  DDS_DataWriterQos & qos )
```

Sets the quality of service of the request DataWriter.

See also

qos_profile (p. 2180)

9.370.3.6 datareader_qos()

```
RequesterParams & connext::RequesterParams::datareader_qos (
            const  DDS_DataReaderQos & qos )
```

Sets the quality of service of the request DataReader.

See also

qos_profile (p. 2180)

9.370.3.7 publisher()

```
RequesterParams & connext::RequesterParams::publisher (
            DDSPublisher * publisher )
```

Sets a specific Publisher.

By default, a **Requester** (p. 2157) uses the DomainParticipant's implicit Publisher. Sometimes a different Publisher may be needed, for example, to use non-default PublisherQos.

9.370.3.8 subscriber()

```
RequesterParams & connext::RequesterParams::subscriber (
            DDSSubscriber * subscriber )
```

Sets a specific Subscriber.

By default, a **Requester** (p. 2157) uses the DomainParticipant's implicit Subscriber. Sometimes a different Subscriber may be needed, for example, to use non-default SubscriberQos.

9.370.3.9 request_type_support()

```
RequesterParams & connext::RequesterParams::request_type_support (
            TypeSupport * type_support )
```

Sets the type support for the request type.

This parameter is required when `TReq` is **DDS_DynamicData** (p. 1013).

In all other cases, this parameter is ignored and the singleton **Requester<TReq, TRep>::RequestTypeSupport** (p. 2161) is used instead.

See also

DDSDynamicDataTypeSupport (p. 1725)

9.370.3.10 reply_type_support()

```
RequesterParams & connext::RequesterParams::reply_type_support (
            TypeSupport * type_support )
```

Sets the type support for the reply type.

This parameter is required when `TRep` is **DDS_DynamicData** (p. 1013).

In all other cases, this parameter is ignored and the singleton **Requester<TReq, TRep>::ReplyTypeSupport** (p. 2161) is used instead.

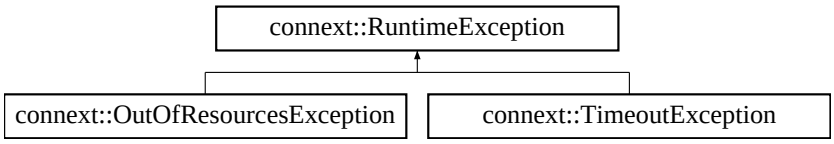
See also

DDSDynamicDataTypeSupport (p. 1725)

9.371 connext::RuntimeException Class Reference

Generic, unspecified error.

Inheritance diagram for connext::RuntimeException:



9.371.1 Detailed Description

Generic, unspecified error.

Its base class is `std::runtime_error`, from which it inherits the `what()` method.

See also

DDS_RETCODE_ERROR (p. 532)

9.372 connext::Sample< T > Class Template Reference

A data sample and related info received from the middleware.

Inherits connext::details::SampleBase< T, I >.

Public Member Functions

- **Sample** ()
  Creates a sample with default data and info values.
- **Sample** (const **Sample**< T > &other)
  Creates a sample making a deep copy of another sample.
- **Sample** (**SampleRef**< T > sample_ref)
  *Creates a sample as a deep copy of the contents referenced by a **WriteSampleRef** (p. 2223).*
- **SampleIdentity_t identity** () const
  Gets the identity of this sample.
- **SampleIdentity_t related_identity** () const
  Gets the identity of a sample that is related to this one.
- T & **data** ()
  Gets the data this sample contains.
- **SampleInfo** & **info** ()
  Gets the sample information.
- const T & **data** ()
  Gets the data this sample contains.
- const **SampleInfo** & **info** ()
  Gets the sample information.

9.372.1 Detailed Description

template<typename T>
class connext::Sample< T >

A data sample and related info received from the middleware.

Samples contain data received from the middleware and information associated to the data.

Template Parameters

T	The data type that this sample contains

See also

connext::Requester::receive_reply(Sample<TRep>&, const Duration_t&) (p. 2164)
connext::Requester::take_reply(Sample<TRep>&) (p. 2169)
connext::Replier::receive_request(Sample<TReq>&, const Duration_t&) (p. 2146)

9.372.2 Constructor & Destructor Documentation

9.372.2.1 Sample() [1/3]

```
template<typename T >
connext::Sample< T >::Sample ( )  [inline]
```

Creates a sample with default data and info values.

The data is deeply initialized using the corresponding TypeSupport

See also

FooTypeSupport::initialize_data (p. 1986)

9.372.2.2 Sample() [2/3]

```
template<typename T >
connext::Sample< T >::Sample (
            const  Sample< T > & other )  [inline]
```

Creates a sample making a deep copy of another sample.

9.372.2.3 Sample() [3/3]

```
template<typename T >
connext::Sample< T >::Sample (
            SampleRef< T > sample_ref )
```

Creates a sample as a deep copy of the contents referenced by a **WriteSampleRef** (p. 2223).

9.372.3 Member Function Documentation

9.372.3.1 identity()

```
template<typename T >
SampleIdentity_t  connext::Sample< T >::identity ( ) const  [inline]
```

Gets the identity of this sample.

The identity is assigned by the middleware upon reception.

See also

Correlating requests and replies (p. 423)

References **connext::Sample< T >::info()**.

9.372.3.2 related_identity()

```
template<typename T >
SampleIdentity_t  connext::Sample< T >::related_identity ( ) const  [inline]
```

Gets the identity of a sample that is related to this one.

When a **connext::Requester** (p. 2157) receives a reply, the reply **Sample** (p. 2183) contains the identity of the related request.

Returns

The identity of another sample that is related to this one, or **DDS_UNKNOWN_SAMPLE_IDENTITY** (p. 676) if there is not a related sample.

See also

connext::Requester::receive_reply(Sample<TRep>&, const Duration_t&) (p. 2164)

Correlating requests and replies (p. 423)

References **connext::Sample< T >::info()**.

9.372.3.3 data() [1/2]

```
template<typename T >
T &  connext::Sample< T >::data ( )
```

Gets the data this sample contains.

See also

Basic Requester example (p. 421)

9.372.3.4 info() [1/2]

```
template<typename T >
SampleInfo &  connext::Sample< T >::info ( )
```

Gets the sample information.

See also

DDS_SampleInfo (p. 1344)

Basic Requester example (p. 421)

Referenced by **connext::Sample< T >::identity()**, and **connext::Sample< T >::related_identity()**.

9.372.3.5 data() [2/2]

```
template<typename T >
const T &  connext::Sample< T >::data ( )
```

Gets the data this sample contains.

See also

Basic Requester example (p. 421)

9.372.3.6 info() [2/2]

```
template<typename T >
const  SampleInfo &  connext::Sample< T >::info ( )
```

Gets the sample information.

See also

DDS_SampleInfo (p. 1344)

Basic Requester example (p. 421)

9.373 rti::flat::Sample< OffsetType > Class Template Reference

The generic definition of FlatData topic-types.

`#include <FlatSample.hpp>`

Inherits rti::flat::SampleBase.

Public Types

- typedef OffsetType **Offset**
  The related Offset type.
- typedef OffsetType::ConstOffset **ConstOffset**
  The related read-only Offset type.

Public Member Functions

- **Offset root** ()
  Provides the Offset to the top-level type.
- **ConstOffset root** () const
  Provides the Offset to the top-level type.
- **Sample**< OffsetType > * **clone** () const
  *Clones a **Sample** (p. 2187), creating an unmanaged **Sample** (p. 2187).*

Static Public Member Functions

- static **Sample**< OffsetType > * **create_data** ()
  *Creates an unmanaged FlatData **Sample** (p. 2187).*
- static void **delete_data** (**rti::flat::Sample**< OffsetType > *sample)
  *Releases an unmanaged **Sample** (p. 2187).*

9.373.1 Detailed Description

template<typename OffsetType>
class rti::flat::Sample< OffsetType >

The generic definition of FlatData topic-types.

Template Parameters

OffsetType	The Offset to the beginning of the data **Sample** (p. 2187) (the root), for example **MyFlatFinalOffset** (p. 2014), or **MyFlatMutableOffset** (p. 2025).

For a FlatData IDL type, **rtiddsgen** generates an instantiation of this class, such as MyFlatFinal or MyFlatMutable. That's the **topic-type** used to declare a **DDSTopic** (p. 1885) and write or read FlatData samples.

A FlatData **Sample** (p. 2187) owns an inline buffer that contains the serialized data. To access the **Sample** (p. 2187) data members we use **Offsets** (p. 771), iterators that point to the position of a member in the **Sample** (p. 2187) buffer. The **root()** (p. 2189) is the Offset to the top-level element (for example **MyFlatMutableOffset** (p. 2025)). From there **MyFlatMutableOffset** (p. 2025)'s member functions provide access to its members.

The method to create a **Sample** (p. 2187) varies depending on whether the type is final or mutable.

Samples of **final** FlatData types always have the same size, and can be obtained directly with **FooDataWriter::get_loan()** (p. 1963).

Samples of **mutable** FlatData types need to be "built" using the Builder returned by **rti::flat::build_data()** (p. 768) (**MyFlatMutableBuilder** (p. 2018) in this example). A Builder allows adding each member individually in any order, and sizing sequences as needed.

See **Publishing FlatData** (p. 777) for code snippets.

Samples created with **FooDataWriter::get_loan()** (p. 1963) or **rti::flat::build_data()** (p. 768) are DataWriter-managed and do not need to be explicitly deleted. See **FooDataWriter::get_loan()** (p. 1963) for an explanation of the lifecycle of DataWriter-managed samples.

Samples received with a **FooDataReader** (p. 1917) can be examined by obtaining the **root()** (p. 2189), but cannot be modified. See **Subscribing to FlatData** (p. 778).

FlatData samples are not value types. They don't provide copy constructors or assignment operators, because of their definition as an inline buffer that contains the serialized **Sample** (p. 2187) at all times. A **Sample** (p. 2187) can be cloned with the static function **clone()** (p. 2190). A cloned **Sample** (p. 2187) is unmanaged and has to be deleted with the static function **delete_data()** (p. 2190).

9.373.2 Member Typedef Documentation

9.373.2.1 Offset

```
template<typename OffsetType >
typedef OffsetType  rti::flat::Sample< OffsetType >::Offset
```

The related Offset type.

This is the template parameter, OffsetType

9.373.2.2 ConstOffset

```
template<typename OffsetType >
typedef OffsetType::ConstOffset  rti::flat::Sample< OffsetType >::ConstOffset
```

The related read-only Offset type.

For example, **MyFlatMutableOffset::ConstOffset** (p. 2026).

9.373.3 Member Function Documentation

9.373.3.1 root() [1/2]

```
template<typename OffsetType >
Offset  rti::flat::Sample< OffsetType >::root ( )  [inline]
```

Provides the Offset to the top-level type.

Returns

An Offset that allows reading and modifying the members of this **Sample** (p. 2187), for example **MyFlatFinalOffset** (p. 2014) or **MyFlatMutableOffset** (p. 2025).

This overload allows modifying the **Sample** (p. 2187).

Example:

```
MyFlatFinal *my_sample = ...; // for example, created with DataWriter::get_loan()
MyFlatFinalOffset my_sample_root = my_sample->root();
my_sample_root.my_primitive(33);
auto my_member_offset = my_sample_root.my_complex();
// ... modify my_member_offset
```

9.373.3.2 root() [2/2]

```
template<typename OffsetType >
ConstOffset  rti::flat::Sample< OffsetType >::root ( ) const  [inline]
```

Provides the Offset to the top-level type.

Returns

An Offset that allows reading the members of this **Sample** (p. 2187). For example, **MyFlatFinalOffset::ConstOffset** (p. 2015) or **MyFlatMutableOffset::ConstOffset** (p. 2026).

This overload doesn't allow modifying the **Sample** (p. 2187).

Example:

```
const MyFlatMutable *my_sample = ...; // for example, read from a DataReader
MyFlatMutableOffset::ConstOffset my_sample_root = my_sample->root();
std::cout « my_sample_root.my_primitive() « std::endl;
auto my_member_offset = my_sample_root.my_mutable_seq();
// ... read my_member_offset
```

9.373.3.3 create_data()

```
template<typename OffsetType >
Sample< OffsetType > *  rti::flat::Sample< OffsetType >::create_data  [static]
```

Creates an unmanaged FlatData **Sample** (p. 2187).

Note

In general on the publication side it is recommended to create DataWriter-managed samples using **FooData↩ Writer::get_loan** (p. 1963) (for final types) and **rti::flat::build_data()** (p. 768) (for mutable types).

If the topic-type is final, the returned **Sample** (p. 2187) can be initialized by obtaining the **root()** (p. 2189).

If the topic-type is mutable, the resulting **Sample** (p. 2187) has no members, and **root()** (p. 2189) returns a **null** (p. 2128) Offset. The result of **create_data()** (p. 2189) must be used to create a Builder. This is a rare use case, and **rti::flat↩ ::build_data()** (p. 768) should be used in most situations.

In all cases, the returned **Sample** (p. 2187) must be explicitly deleted with **delete_data()** (p. 2190).

9.373.3.4 clone()

```
template<typename OffsetType >
Sample< OffsetType > *  rti::flat::Sample< OffsetType >::clone ( ) const  [inline]
```

Clones a **Sample** (p. 2187), creating an unmanaged **Sample** (p. 2187).

Warning

When this **Sample** (p. 2187) has been created with **FooDataWriter::get_loan** (p. 1963) or **rti::flat::build_data()** (p. 768), the clone this function creates cannot be used in **FooDataWriter::write** (p. 1951), because the DataWriter is managing the lifecycle of its samples.

Creates a new **Sample** (p. 2187) and copies the underlying serialized buffer.

delete_data() (p. 2190) must be called to release the **Sample** (p. 2187).

9.373.3.5 delete_data()

```
template<typename OffsetType >
static void  rti::flat::Sample< OffsetType >::delete_data (
              rti::flat::Sample< OffsetType > * sample )  [inline], [static]
```

Releases an unmanaged **Sample** (p. 2187).

Unmanaged samples are those created with **create_data()** (p. 2189) or **clone()** (p. 2190).

Precondition

`sample` cannot have been created with **FooDataWriter::get_loan** (p. 1963) or **rti::flat::build_data()** (p. 768).

Postcondition

`sample` becomes invalid and shouldn't be used

9.374 connext::SampleIterator< T, IsConst > Class Template Reference

STL-compliant random-access iterator for SampleRef<T>

Public Member Functions

- **SampleIterator** & **operator++** ()

9.374.1 Detailed Description

template<typename T, bool IsConst>
class connext::SampleIterator< T, IsConst >

STL-compliant random-access iterator for SampleRef<T>

See also

connext::LoanedSamples::begin (p. 2003)

9.374.2 Member Function Documentation

9.374.2.1 operator++()

```
template<typename T , bool IsConst>
SampleIterator &  connext::SampleIterator< T, IsConst >::operator++ ( )  [inline]
```

Coverity complains about this increment because it may overflow. However the user is responsible for not incrementing the iterator beyond the end of the sequence when accessing the iterator. With the current implementation we are providing a big enough iterator to iterate over a full sequence without overflowing.

9.375 connext::SampleRef< T > Class Template Reference

A data sample and related information received from the middleware.

Public Member Functions

- **SampleRef** ()
  Creates an object with no data.
- **SampleRef** (T * **data**, Info * **info**)
  Creates an object with a reference to a data object.
- **SampleRef** (T & **data**, Info & **info**)
  Creates an object with a reference to a data object.
- **SampleRef** (**Sample**< T > &sample)
  Copies the references to data and info.
- **SampleRef** & **operator=** (**Sample**< T > &sample)
  Copies the references to data and info.
- T & **data** () const
  Gets the data this sample contains.
- Info & **info** () const
  Gets the SampleInfo.
- **operator T&** () const
  Gets the data this sample contains.
- void **set_data** (T & **data**)
  Sets a reference to a data object.
- void **set_info** (**DDS_SampleInfo** & **info**)
  Sets the info object.
- bool **is_nil_data** () const
  *Indicates if this **SampleRef** (p. 2191) contains a reference to data or not.*
- bool **is_nil_info** () const
  *Indicates if this **SampleRef** (p. 2191) contains a reference to data or not.*
- **SampleIdentity_t** **identity** () const
  Gets the identity of this sample.
- **SampleIdentity_t** **related_identity** () const
  Gets the identity of a sample that is related to this one.

9.375.1 Detailed Description

template<**typename T**>
class connext::SampleRef< **T** >

A data sample and related information received from the middleware.

A **SampleRef** (p. 2191) can be used much like a **connext::Sample** (p. 2183), but it's different in that it holds references to data and **DDS_SampleInfo** (p. 1344), whereas a **Sample** (p. 2183) is a value type and owns its data and info.

When calling an operation that copies data from the middleware into a **Sample** (p. 2183) (e.g. **connext::Requester**↩ **::take_reply(Sample**<**TRep**>**&)** (p. 2169)), the references to its data and information must have been set explicitly (see **connext::SampleRef::set_data(T&)** (p. 2196) and **connext::SampleRef::set_info(DDS_SampleInfo&)** (p. 2196))

When a **SampleRef** (p. 2191) is copied, the destination **SampleRef** (p. 2191) references the same data and info objects as the source. Copying and passing by value are inexpensive operations.

SampleRef (p. 2191) is also the type of the elements in a **connext::LoanedSamples** (p. 1995) container and they hold references to data from the middleware.

Template Parameters

T	The data type that this sample contains

See also

connext::Requester::receive_reply(Sample<TRep>&, const Duration_t&) (p. 2164)

connext::Requester::take_reply(Sample<TRep>&) (p. 2169)

connext::Replier::receive_request(Sample<TReq>&, const Duration_t&) (p. 2146)

Basic Requester example using SampleRef (p. 424)

Template Parameters

T	The data type that this sample can contain

9.375.2 Constructor & Destructor Documentation

9.375.2.1 SampleRef() [1/4]

```
template<typename T >
connext::SampleRef< T >::SampleRef ( )  [inline]
```

Creates an object with no data.

See also

set_data(T&) (p. 2196)

Referenced by **connext::SampleRef< T >::operator=()**.

9.375.2.2 SampleRef() [2/4]

```
template<typename T >
connext::SampleRef< T >::SampleRef (
            T * data,
            Info * info )  [inline]
```

Creates an object with a reference to a data object.

See also

set_data(T&) (p. 2196)

9.375.2.3 SampleRef() [3/4]

```
template<typename T >
connext::SampleRef< T >::SampleRef (
            T & data,
            Info & info )  [inline]
```

Creates an object with a reference to a data object.

See also

set_data(T&) (p. 2196)

9.375.2.4 SampleRef() [4/4]

```
template<typename T >
connext::SampleRef< T >::SampleRef (
             Sample< T > & sample )  [inline]
```

Copies the references to data and info.

After this operation, this object contains a reference to the same data and info instances as the source.

9.375.3 Member Function Documentation

9.375.3.1 operator=()

```
template<typename T >
SampleRef &  connext::SampleRef< T >::operator= (
             Sample< T > & sample )  [inline]
```

Copies the references to data and info.

After this operation, this object contains a reference to the same data and info instances as the source.

References **connext::SampleRef< T >::SampleRef()**.

9.375.3.2 data()

```
template<typename T >
T &  connext::SampleRef< T >::data ( ) const  [inline]
```

Gets the data this sample contains.

Precondition

Data must have been set before

See also

connext::WriteSampleRef::set_data (p. 2225)

connext::WriteSample::data() (p. 2221)

Referenced by **connext::SampleRef< T >::set_data()**.

9.375.3.3 info()

```
template<typename T >
Info &  connext::SampleRef< T >::info ( ) const  [inline]
```

Gets the SampleInfo.

Precondition

The info object must have been set

See also

set_info(DDS_SampleInfo&) (p. 2196)

connext::SampleRef::identity() (p. 2197)

Referenced by **connext::SampleRef< T >::identity()**, **connext::SampleRef< T >::related_identity()**, and **connext::SampleRef< T >::set_info()**.

9.375.3.4 operator T&()

```
template<typename T >
connext::SampleRef< T >::operator T& ( ) const  [inline]
```

Gets the data this sample contains.

Precondition

Data must have been set before

See also

connext::WriteSampleRef::set_data (p. 2225)

connext::WriteSample::data() (p. 2221)

9.375.3.5 set_data()

```
template<typename T >
void  connext::SampleRef< T >::set_data (
            T & data )  [inline]
```

Sets a reference to a data object.

Parameters

data	The data object this sample will reference

References **connext::SampleRef< T >::data()**.

9.375.3.6 set_info()

```
template<typename T >
void  connext::SampleRef< T >::set_info (
             DDS_SampleInfo & info )  [inline]
```

Sets the info object.

References **connext::SampleRef< T >::info()**.

9.375.3.7 is_nil_data()

```
template<typename T >
bool  connext::SampleRef< T >::is_nil_data ( ) const  [inline]
```

Indicates if this **SampleRef** (p. 2191) contains a reference to data or not.

Returns

True if this sample does not contain a reference to data.

See also

connext::SampleRef::data (p. 2194)
connext::SampleRef::set_data(T&) (p. 2196)

9.375.3.8 is_nil_info()

```
template<typename T >
bool  connext::SampleRef< T >::is_nil_info ( ) const  [inline]
```

Indicates if this **SampleRef** (p. 2191) contains a reference to data or not.

Returns

True if this sample does not contain a reference to info.

See also

connext::SampleRef::info() (p. 2195)
connext::SampleRef::set_info(DDS_SampleInfo&) (p. 2196)

9.375.3.9 identity()

```
template<typename T >
SampleIdentity_t  connext::SampleRef< T >::identity ( ) const  [inline]
```

Gets the identity of this sample.

Precondition

The info object must have been set

See also

set_info(DDS_SampleInfo&) (p. 2196)
connext::Sample::identity() (p. 2185)

References **connext::SampleRef< T >::info()**.

9.375.3.10 related_identity()

```
template<typename T >
SampleIdentity_t  connext::SampleRef< T >::related_identity ( ) const  [inline]
```

Gets the identity of a sample that is related to this one.

Precondition

The info object must have been set

See also

connext::Sample::related_identity() (p. 2185)

References **connext::SampleRef< T >::info()**.

9.376 rti::flat::SequenceIterator< E, OffsetKind > Class Template Reference

Iterator for collections of Offsets.

```
#include <SequenceIterator.hpp>
```

Public Types

- typedef std::forward_iterator_tag **iterator_category**

  The iterator category.
- typedef E **value_type**

  The element type.
- typedef **value_type** **reference**

  The reference type is the same as the value type, an Offset.
- typedef **value_type** **pointer**

  The pointer type is the same as the value type, an Offset.
- typedef std::ptrdiff_t **difference_type**

  The difference type.

Public Member Functions

- **SequenceIterator** ()

  Constructs an invalid iterator.
- bool **is_null** () const

  Returns whether the iterator is invalid.
- **value_type operator*** () const

  Returns the Offset of the current element.
- **value_type operator->** () const

  Returns the Offset of the current element.
- bool **advance** ()

  Advances to the next element, reporting any errors by returning false.
- **SequenceIterator** & **operator++** ()

  Advances to the next element.
- **SequenceIterator operator++** (int)

  Advances to the next element.

Friends

- bool **operator<** (const **SequenceIterator** &s1, const **SequenceIterator** &s2)

  Compares two iterators.
- bool **operator>** (const **SequenceIterator** &s1, const **SequenceIterator** &s2)

  Compares two iterators.
- bool **operator<=** (const **SequenceIterator** &s1, const **SequenceIterator** &s2)

  Compares two iterators.
- bool **operator>=** (const **SequenceIterator** &s1, const **SequenceIterator** &s2)

  Compares two iterators.
- bool **operator==** (const **SequenceIterator** &s1, const **SequenceIterator** &s2)

  Compares two iterators.
- bool **operator!=** (const **SequenceIterator** &s1, const **SequenceIterator** &s2)

  Compares two iterators.

9.376.1 Detailed Description

template<typename E, typename OffsetKind>
class rti::flat::SequenceIterator< E, OffsetKind >

Iterator for collections of Offsets.

Template Parameters

E	The Offset type of the elements
OffsetKind	(implementation detail)

Note

This type should not be declared directly. For example, for a **SequenceOffset** (p. 2204), use **SequenceOffset**↩ **::iterator** (p. 786) (or simply `auto`).

A **SequenceIterator** (p. 2198) moves through the Offset to elements of a sequence or array **Offset** (p. 771), and provides the functionality of a standard `forward_iterator`.

Note that a dereferenced **SequenceIterator** (p. 2198) returns **by value** the Offset to the current element; it doesn't return it by reference.

Since the Traditional C++ API doesn't use exceptions, it is recommended to use the function **advance()** (p. 2202) instead of operator++ to check for errors

9.376.2 Member Typedef Documentation

9.376.2.1 iterator_category

```
template<typename E , typename OffsetKind >
typedef std::forward_iterator_tag  rti::flat::SequenceIterator< E, OffsetKind >::iterator_↩
category
```

The iterator category.

9.376.2.2 value_type

```
template<typename E , typename OffsetKind >
typedef E  rti::flat::SequenceIterator< E, OffsetKind >::value_type
```

The element type.

9.376.2.3 reference

```
template<typename E , typename OffsetKind >
typedef  value_type  rti::flat::SequenceIterator< E, OffsetKind >::reference
```

The reference type is the same as the value type, an Offset.

9.376.2.4 pointer

```
template<typename E , typename OffsetKind >
typedef  value_type  rti::flat::SequenceIterator< E, OffsetKind >::pointer
```

The pointer type is the same as the value type, an Offset.

9.376.2.5 difference_type

```
template<typename E , typename OffsetKind >
typedef std::ptrdiff_t  rti::flat::SequenceIterator< E, OffsetKind >::difference_type
```

The difference type.

9.376.3 Constructor & Destructor Documentation

9.376.3.1 SequenceIterator()

```
template<typename E , typename OffsetKind >
rti::flat::SequenceIterator< E, OffsetKind >::SequenceIterator ( )  [inline]
```

Constructs an invalid iterator.

9.376.4 Member Function Documentation

9.376.4.1 is_null()

```
template<typename E , typename OffsetKind >
bool  rti::flat::SequenceIterator< E, OffsetKind >::is_null ( ) const  [inline]
```

Returns whether the iterator is invalid.

9.376.4.2 operator*()

```
template<typename E , typename OffsetKind >
value_type  rti::flat::SequenceIterator< E, OffsetKind >::operator* ( ) const  [inline]
```

Returns the Offset of the current element.

Returns

The Offset to the current element. Note that this function returns by value, not by reference. This Offset may be **null** (p. 2128) if the iterator has surpassed the length of the collection. See **Offset Error Management** (p. 773).

9.376.4.3 operator->()

```
template<typename E , typename OffsetKind >
value_type  rti::flat::SequenceIterator< E, OffsetKind >::operator-> ( ) const  [inline]
```

Returns the Offset of the current element.

9.376.4.4 advance()

```
template<typename E , typename OffsetKind >
bool  rti::flat::SequenceIterator< E, OffsetKind >::advance ( )  [inline]
```

Advances to the next element, reporting any errors by returning false.

Since the Traditional C++ API doesn't support exceptions, this function is recommended instead of operator++.

Returns

True if the function succeeds, or false if there was an error

9.376.4.5 operator++() [1/2]

```
template<typename E , typename OffsetKind >
SequenceIterator &  rti::flat::SequenceIterator< E, OffsetKind >::operator++ ( )  [inline]
```

Advances to the next element.

Warning

Since the Traditional C++ API doesn't use exceptions, use **advance()** (p. 2202) instead of this operator to check for errors.

9.376.4.6 operator++() [2/2]

```
template<typename E , typename OffsetKind >
SequenceIterator  rti::flat::SequenceIterator< E, OffsetKind >::operator++ (
            int  )  [inline]
```

Advances to the next element.

Warning

Since the Traditional C++ API doesn't use exceptions, use **advance()** (p. 2202) instead of this operator to check for errors.

9.376.5 Friends And Related Function Documentation

9.376.5.1 operator<

```
template<typename E , typename OffsetKind >
bool operator< (
            const  SequenceIterator< E, OffsetKind > & s1,
            const  SequenceIterator< E, OffsetKind > & s2 )  [friend]
```

Compares two iterators.

9.376.5.2 operator>

```
template<typename E , typename OffsetKind >
bool operator> (
            const  SequenceIterator< E, OffsetKind > & s1,
            const  SequenceIterator< E, OffsetKind > & s2 )  [friend]
```

Compares two iterators.

9.376.5.3 operator<=

```
template<typename E , typename OffsetKind >
bool operator<= (
            const  SequenceIterator< E, OffsetKind > & s1,
            const  SequenceIterator< E, OffsetKind > & s2 )  [friend]
```

Compares two iterators.

9.376.5.4 operator>=

```
template<typename E , typename OffsetKind >
bool operator>= (
            const  SequenceIterator< E, OffsetKind > & s1,
            const  SequenceIterator< E, OffsetKind > & s2 )  [friend]
```

Compares two iterators.

9.376.5.5 operator==

```
template<typename E , typename OffsetKind >
bool operator== (
            const  SequenceIterator< E, OffsetKind > & s1,
            const  SequenceIterator< E, OffsetKind > & s2 )  [friend]
```

Compares two iterators.

9.376.5.6 operator"!=

```
template<typename E , typename OffsetKind >
bool operator!= (
            const  SequenceIterator< E, OffsetKind > & s1,
            const  SequenceIterator< E, OffsetKind > & s2 )  [friend]
```

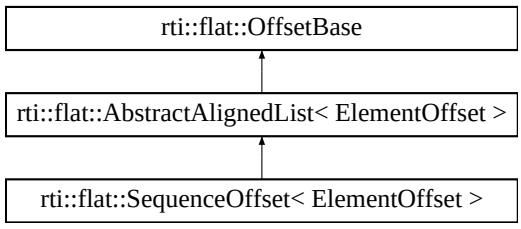
Compares two iterators.

9.377 rti::flat::SequenceOffset< ElementOffset > Class Template Reference

Offset to a sequence of non-primitive elements.

`#include <SequenceOffsets.hpp>`

Inheritance diagram for rti::flat::SequenceOffset< ElementOffset >:



Public Member Functions

- ElementOffset **get_element** (unsigned int i)

  Gets the Offset to an element.
- unsigned int **element_count** () const

  The number of elements.

Additional Inherited Members

9.377.1 Detailed Description

template<typename ElementOffset>
class rti::flat::SequenceOffset< ElementOffset >

Offset to a sequence of non-primitive elements.

Template Parameters

ElementOffset	The Offset type, for example **MyFlatMutableOffset** (p. 2025), **MyFlatFinalOffset** (p. 2014), or **StringOffset** (p. 2216).

Represents an Offset to a sequence member and allows getting an Offset to each of its elements.

A **SequenceOffset** (p. 2204) may meet the requirements to be cast to an array of the equivalent plain C++ element type (see **rti::flat::plain_cast** (p. 773)), if the `ElementOffset` is a final type.

9.377.2 Member Function Documentation

9.377.2.1 get_element()

```
template<typename ElementOffset >
ElementOffset  rti::flat::SequenceOffset< ElementOffset >::get_element (
             unsigned int i )  [inline]
```

Gets the Offset to an element.

Parameters

i	The zero-based index to the element

Returns

The Offset to the element in the i-th position

9.377.2.2 element_count()

```
template<typename ElementOffset >
unsigned int  rti::flat::SequenceOffset< ElementOffset >::element_count ( ) const  [inline]
```

The number of elements.

9.378 connext::SimpleReplier< TReq, TRep > Class Template Reference

A callback-based replier.

Public Member Functions

- **SimpleReplier** (**DDSDomainParticipant** *participant, const std::string &service_name, **SimpleReplier**↩ **Listener**< TReq, TRep > &listener)

  *Creates a new **SimpleReplier** (p. 2206).*
- **SimpleReplier** (const **SimpleReplierParams**< TReq, TRep > &params)

  *Creates a new **SimpleReplier** (p. 2206).*
- virtual ~**SimpleReplier** ()

  *Releases the resources created by this **SimpleReplier** (p. 2206).*

9.378.1 Detailed Description

template<typename TReq, typename TRep>
class connext::SimpleReplier< TReq, TRep >

A callback-based replier.

A **SimpleReplier** (p. 2206) is based on a **connext::SimpleReplierListener** (p. 2208) that users provide . Requests are passed to the callback, which returns a reply. The reply is directed only to the **Requester** (p. 2157) that sent the request.

SimpleRepliers are useful for simple use cases where a single reply for a request can be generated quickly, for example, looking up a table.

When more than one reply for a request can be generated or the processing is complex or needs to happen asynchronously, use a **connext::Replier** (p. 2139) instead.

See also

connext::Replier (p. 2139)

connext::SimpleReplierListener (p. 2208)

SimpleReplier example (p. 426)

9.378.2 Constructor & Destructor Documentation

9.378.2.1 SimpleReplier() [1/2]

```
template<typename TReq , typename TRep >
connext::SimpleReplier< TReq, TRep >::SimpleReplier (
             DDSDomainParticipant * participant,
            const std::string & service_name,
             SimpleReplierListener< TReq, TRep > & listener )
```

Creates a new **SimpleReplier** (p. 2206).

See also

connext::SimpleReplierParams (p. 2209)

connext::SimpleReplierListener (p. 2208)

9.378.2.2 SimpleReplier() [2/2]

```
template<typename TReq , typename TRep >
connext::SimpleReplier< TReq, TRep >::SimpleReplier (
            const  SimpleReplierParams< TReq, TRep > & params )  [explicit]
```

Creates a new **SimpleReplier** (p. 2206).

See also

connext::SimpleReplierParams (p. 2209)

connext::SimpleReplierListener (p. 2208)

9.378.2.3 ~SimpleReplier()

```
template<typename TReq , typename TRep >
connext::SimpleReplier< TReq, TRep >::~ SimpleReplier  [virtual]
```

Releases the resources created by this **SimpleReplier** (p. 2206).

See also

connext::Replier::~Replier() (p. 2144)

9.379 connext::SimpleReplierListener< TReq, TRep > Class Template Reference

The listener called by a **SimpleReplier** (p. 2206).

Public Member Functions

- virtual TRep * **on_request_available** (typename dds_type_traits< TReq >::SampleRefType request)=0

  User callback that receives a request and provides a reply.
- virtual void **return_loan** (TRep *reply)=0

  Returns a previously generated reply to the user.

9.379.1 Detailed Description

template<class TReq, class TRep>
class connext::SimpleReplierListener< TReq, TRep >

The listener called by a **SimpleReplier** (p. 2206).

See also

connext::SimpleReplier (p. 2206)

SimpleReplier example (p. 426)

9.379.2 Member Function Documentation

9.379.2.1 on_request_available()

```
template<class TReq , class TRep >
virtual TRep *  connext::SimpleReplierListener< TReq, TRep >::on_request_available (
             typename dds_type_traits< TReq >::SampleRefType request )  [pure virtual]
```

User callback that receives a request and provides a reply.

This operation gets called when a request is available and expects a reply that is automatically sent. Immediately after that, **return_loan** (p. 2209) is called.

Parameters

request	The received request

Returns

A reply for that request

9.379.2.2 return_loan()

```
template<class TReq , class TRep >
virtual void  connext::SimpleReplierListener< TReq, TRep >::return_loan (
            TRep * reply )  [pure virtual]
```

Returns a previously generated reply to the user.

This operation is always called right after sending the reply created by **on_request_available** (p. 2208). It can be used to release any resources from the reply creation. If there are no resources to release, the implementation body can be empty.

Parameters

reply	The reply previously provided in **on_request_available** (p. 2208)

9.380 connext::SimpleReplierParams< TReq, TRep > Class Template Reference

Contains the parameters for creating a **connext::SimpleReplier** (p. 2206).

Inherits connext::details::EntityParams.

Public Member Functions

- **SimpleReplierParams** (**DDS::DomainParticipant** *participant, **SimpleReplierListener**< TReq, TRep > &listener)

  *Creates **SimpleReplierParams** (p. 2209) with the parameters that a **SimpleReplier** (p. 2206) always needs.*
- **SimpleReplierParams** & **service_name** (const std::string &service_name)

  *The service name the **Replier** (p. 2139) offers and Requesters use to match.*
- **SimpleReplierParams** & **request_topic_name** (const std::string &req_topic)

  Sets a specific request topic name.
- **SimpleReplierParams** & **reply_topic_name** (const std::string &rep_topic)

  Sets a specific reply topic name.
- **SimpleReplierParams** & **qos_profile** (const std::string &qos_library_name, const std::string &qos_profile_↩name)

  Sets a QoS profile for the entities in this replier.
- **SimpleReplierParams** & **datawriter_qos** (const **DDS_DataWriterQos** &qos)

Sets the quality of service of the reply DataWriter.

- **SimpleReplierParams** & **datareader_qos** (const **DDS_DataReaderQos** &qos)

  Sets the quality of service of the request DataReader.
- **SimpleReplierParams** & **publisher** (**DDSPublisher** *publisher)

  Sets a specific Publisher.
- **SimpleReplierParams** & **subscriber** (**DDSSubscriber** *subscriber)

  Sets a specific Subscriber.
- **SimpleReplierParams** & **request_type_support** (**DDS::TypeSupport** *type_support)

  Sets the type support for the request type.
- **SimpleReplierParams** & **reply_type_support** (**DDS::TypeSupport** *type_support)

  Sets the type support for the reply type.

9.380.1 Detailed Description

template<typename TReq, typename TRep>
class connext::SimpleReplierParams< TReq, TRep >

Contains the parameters for creating a **connext::SimpleReplier** (p. 2206).

The parameters for a **SimpleReplier** (p. 2206) are identical to those of the **Replier** (p. 2139), except for the **SimpleReplierListener** (p. 2208), which is required and has a different user callback.

See also

connext::ReplierParams (p. 2152)

9.380.2 Constructor & Destructor Documentation

9.380.2.1 SimpleReplierParams()

```
template<typename TReq , typename TRep >
connext::SimpleReplierParams< TReq, TRep >::SimpleReplierParams (
            DDS::DomainParticipant * participant,
            SimpleReplierListener< TReq, TRep > & listener )  [inline]
```

Creates **SimpleReplierParams** (p. 2209) with the parameters that a **SimpleReplier** (p. 2206) always needs.

In addition to these parameters, a **SimpleReplier** (p. 2206) needs either:

- A service name (**service_name** (p. 2211)), or
- Custom topic names (**reply_topic_name** (p. 2211) and **request_topic_name** (p. 2211)).

The other parameters are optional.

Parameters

participant	The DomainParticipant that this **SimpleReplier** (p. 2206) uses to join a domain.
listener	The listener where the user callback that a **SimpleReplier** (p. 2206) calls is implemented.

9.380.3 Member Function Documentation

9.380.3.1 service_name()

```
template<typename TReq , typename TRep >
SimpleReplierParams &  connext::SimpleReplierParams< TReq, TRep >::service_name (
            const std::string & service_name )  [inline]
```

The service name the **Replier** (p. 2139) offers and Requesters use to match.

See also

connext::RequesterParams::service_name (p. 2180)

9.380.3.2 request_topic_name()

```
template<typename TReq , typename TRep >
SimpleReplierParams &  connext::SimpleReplierParams< TReq, TRep >::request_topic_name (
            const std::string & req_topic )  [inline]
```

Sets a specific request topic name.

This is an alternative to **service_name** (p. 2211)

9.380.3.3 reply_topic_name()

```
template<typename TReq , typename TRep >
SimpleReplierParams &  connext::SimpleReplierParams< TReq, TRep >::reply_topic_name (
            const std::string & rep_topic )  [inline]
```

Sets a specific reply topic name.

This is an alternative to **service_name** (p. 2211)

9.380.3.4 qos_profile()

```
template<typename TReq , typename TRep >
SimpleReplierParams &  connext::SimpleReplierParams< TReq, TRep >::qos_profile (
            const std::string & qos_library_name,
            const std::string & qos_profile_name )  [inline]
```

Sets a QoS profile for the entities in this replier.

See also

connext::ReplierParams::qos_profile (p. 2154)

9.380.3.5 datawriter_qos()

```
template<typename TReq , typename TRep >
SimpleReplierParams &  connext::SimpleReplierParams< TReq, TRep >::datawriter_qos (
            const  DDS_DataWriterQos & qos )  [inline]
```

Sets the quality of service of the reply DataWriter.

See also

qos_profile (p. 2211)

9.380.3.6 datareader_qos()

```
template<typename TReq , typename TRep >
SimpleReplierParams &  connext::SimpleReplierParams< TReq, TRep >::datareader_qos (
            const  DDS_DataReaderQos & qos )  [inline]
```

Sets the quality of service of the request DataReader.

See also

qos_profile (p. 2211)

9.380.3.7 publisher()

```
template<typename TReq , typename TRep >
SimpleReplierParams & connext::SimpleReplierParams< TReq, TRep >::publisher (
            DDSPublisher * publisher ) [inline]
```

Sets a specific Publisher.

See also

connext::RequesterParams::publisher (p. 2181)

9.380.3.8 subscriber()

```
template<typename TReq , typename TRep >
SimpleReplierParams & connext::SimpleReplierParams< TReq, TRep >::subscriber (
            DDSSubscriber * subscriber ) [inline]
```

Sets a specific Subscriber.

See also

connext::RequesterParams::subscriber (p. 2182)

9.380.3.9 request_type_support()

```
template<typename TReq , typename TRep >
SimpleReplierParams & connext::SimpleReplierParams< TReq, TRep >::request_type_support (
            DDS::TypeSupport * type_support ) [inline]
```

Sets the type support for the request type.

See also

connext::ReplierParams::request_type_support (p. 2156)

9.380.3.10 reply_type_support()

```
template<typename TReq , typename TRep >
SimpleReplierParams & connext::SimpleReplierParams< TReq, TRep >::reply_type_support (
            DDS::TypeSupport * type_support ) [inline]
```

Sets the type support for the reply type.

See also

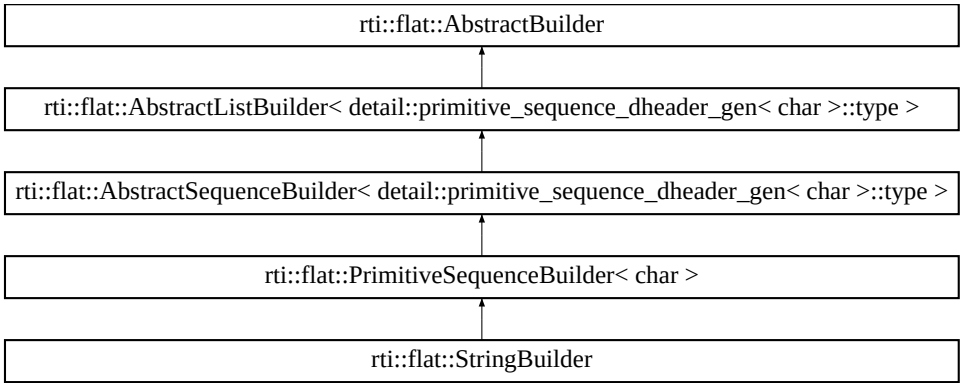
connext::ReplierParams::reply_type_support (p. 2156)

9.381 rti::flat::StringBuilder Class Reference

Builds a string.

`#include <SequenceBuilders.hpp>`

Inheritance diagram for rti::flat::StringBuilder:



Public Member Functions

- **StringBuilder** & **set_string** (const char *value)

  Sets the string value.
- **Offset finish** ()

  Finishes building the string.

Additional Inherited Members

9.381.1 Detailed Description

Builds a string.

A **StringBuilder** (p. 2214) only provides one method, **set_string()** (p. 2215), so it can be typically used as follows:

```
MyFlatMutableBuilder my_builder = ...;
my_builder.build_my_string().set_string("Hello!");
```

Note that by relying on the builder destructor, there is no need to call **finish()** (p. 2215) on the object returned by build↩
_my_string().

9.381.2 Member Function Documentation

9.381.2.1 set_string()

```
StringBuilder & rti::flat::StringBuilder::set_string (
            const char * value )  [inline]
```

Sets the string value.

Parameters

value	The string value to set.

Precondition

value must not be NULL.

References **rti::flat::PrimitiveSequenceBuilder< char >::add_n()**.

9.381.2.2 finish()

```
Offset rti::flat::StringBuilder::finish ( )  [inline]
```

Finishes building the string.

Returns

An Offset to the member that has been built.

See also

discard() (p. 788)

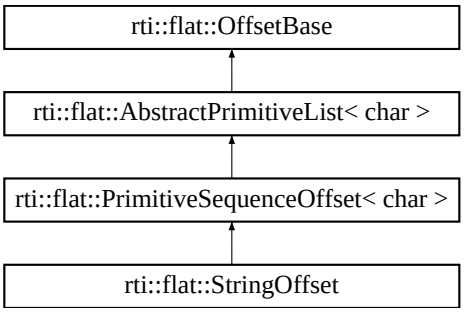
References **rti::flat::PrimitiveSequenceBuilder< char >::add_next()**.

9.382 rti::flat::StringOffset Class Reference

Offset to a string.

`#include <SequenceOffsets.hpp>`

Inheritance diagram for rti::flat::StringOffset:



Public Member Functions

- char * **get_string** ()
  Gets the string.
- const char * **get_string** () const
  Gets the string (const)
- unsigned int **element_count** () const
  Returns the number of characters.

9.382.1 Detailed Description

Offset to a string.

9.382.2 Member Function Documentation

9.382.2.1 get_string() [1/2]

`char * rti::flat::StringOffset::get_string ( ) [inline]`

Gets the string.

The string returned can be modified as long as its size doesn't change.

References **rti::flat::OffsetBase::get_buffer()**.

9.382.2.2 get_string() [2/2]

```
const char * rti::flat::StringOffset::get_string ( ) const  [inline]
```

Gets the string (const)

References **rti::flat::OffsetBase::get_buffer()**.

9.382.2.3 element_count()

```
unsigned int rti::flat::StringOffset::element_count ( ) const  [inline]
```

Returns the number of characters.

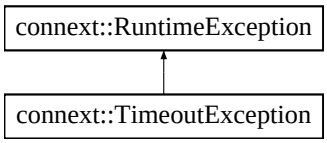
This number doesn't include the null terminating character.

References **rti::flat::PrimitiveSequenceOffset< T >::element_count()**.

9.383 connext::TimeoutException Class Reference

The operation timed out (does not apply to wait or receive operations)

Inheritance diagram for connext::TimeoutException:



9.383.1 Detailed Description

The operation timed out (does not apply to wait or receive operations)

Operations that wait for samples (e.g. **connext::Requester::wait_for_replies(int, const Duration_t&)** (p. 2167) or **connext::Replier::receive_requests(int, int, const Duration_t&)** (p. 2147)) do not throw this exception.

See also

DDS_RETCODE_TIMEOUT (p. 533)

9.384 TransportAllocationSettings_t Struct Reference

Allocation settings used by various internal buffers.

`#include <transport_common_user.h>`

9.384.1 Detailed Description

Allocation settings used by various internal buffers.

An allocation setting structure defines the rules of memory management used by internal buffers.

An internal buffer can provide blocks of memory of fixed size. They are used in several places of any transport, and this structure defines its starting size, limits, and how to increase its capacity.

It contains three values:

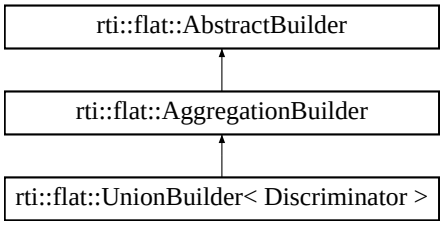
- initial_count: the number of individual elements that are allocated when the buffer is created.
- max_count: the maximum number of elements the buffer can hold. The buffer will grow up to this amount. After this limit is reached, new allocation requests will fail. For unlimited size, use the value NDDS_TRANSPORT_↩ ALLOCATION_SETTINGS_MAX_COUNT_UNLIMITED
- incremental_count: The amount of elements that are allocated at every increment. You can use the value↩ : NDDS_TRANSPORT_ALLOCATION_SETTINGS_INCREMENTAL_COUNT_AUTOMATIC to have the buffer double its size at every reallocation request.

9.385 rti::flat::UnionBuilder< Discriminator > Class Template Reference

Base class of builders for user-defined mutable unions.

`#include <AggregationBuilders.hpp>`

Inheritance diagram for rti::flat::UnionBuilder< Discriminator >:



Additional Inherited Members

9.385.1 Detailed Description

template<typename Discriminator>
class rti::flat::UnionBuilder< Discriminator >

Base class of builders for user-defined mutable unions.

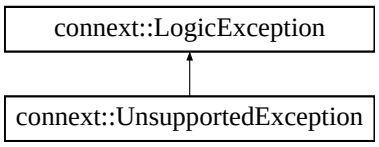
Union builders can only add or build a single member.

This class contains implementation details and doesn't add any public function to **AbstractBuilder** (p. 787).

9.386 connext::UnsupportedException Class Reference

Unsupported operation.

Inheritance diagram for connext::UnsupportedException:



9.386.1 Detailed Description

Unsupported operation.

See also

DDS_RETCODE_UNSUPPORTED (p. 532)

9.387 connext::WriteSample< T > Class Template Reference

A sample for writing data.

Inherits connext::details::SampleBase< T, I >.

Public Member Functions

- **WriteSample** ()

  Creates a sample with default data and info values.
- **WriteSample** (const **WriteSample**< T > &other)

  Creates a sample making a deep copy of another sample.
- **WriteSample** (const T & **data**)

  Creates a sample making a deep copy of another sample.
- **WriteSample** (**WriteSampleRef**< T > wsref)

  Creates a sample deep copying the data and info the source references.
- T & **data** ()

  Gets the data this sample contains.
- **WriteParams_t** & **info** ()

  Gets the write parameters.
- const T & **data** ()

  Gets the data this sample contains.
- const **WriteParams_t** & **info** ()

  Gets the write parameters.
- **SampleIdentity_t** **identity** () const

  Gets the identity of this sample.

9.387.1 Detailed Description

template<typename T>
class connext::WriteSample< T >

A sample for writing data.

WriteSamples are value types containing data that can be sent and optional parameters that configure how the data is written.

The data is automatically initialized using the corresponding **TypeSupport** (p. 1979) for type `T`, and the memory is released in the destructor.

When the data already exists and allocating a new instance is not required, **connext::WriteSampleRef** (p. 2223) can be used instead.

Template Parameters

T	The data type that this sample contains

9.387.2 Constructor & Destructor Documentation

9.387.2.1 WriteSample() [1/4]

```
template<typename T >
connext::WriteSample< T >::WriteSample ( )  [inline]
```

Creates a sample with default data and info values.

The data is deeply initialized using the corresponding TypeSupport

See also

FooTypeSupport::initialize_data (p. 1986)

9.387.2.2 WriteSample() [2/4]

```
template<typename T >
connext::WriteSample< T >::WriteSample (
            const  WriteSample< T > & other )  [inline]
```

Creates a sample making a deep copy of another sample.

9.387.2.3 WriteSample() [3/4]

```
template<typename T >
connext::WriteSample< T >::WriteSample (
            const T & data )
```

Creates a sample making a deep copy of another sample.

9.387.2.4 WriteSample() [4/4]

```
template<typename T >
connext::WriteSample< T >::WriteSample (
             WriteSampleRef< T > wsref )  [inline]
```

Creates a sample deep copying the data and info the source references.

9.387.3 Member Function Documentation

9.387.3.1 data() [1/2]

```
template<typename T >
T &  connext::WriteSample< T >::data ( )
```

Gets the data this sample contains.

9.387.3.2 info() [1/2]

```
template<typename T >
WriteParams_t &  connext::WriteSample< T >::info ( )
```

Gets the write parameters.

The parameters are input and output to write operations. They can be set to configure the write operation (e.g. setting a specific publication timestamp) or they can be looked up afterward (e.g. what was the actual timestamp the sample was written with).

See also

connext::Requester::send_request(WriteSample<URep>&, const SampleIdentity_t&) (p. 2145)

Referenced by **connext::WriteSample< T >::identity()**.

9.387.3.3 data() [2/2]

```
template<typename T >
const T &  connext::WriteSample< T >::data ( )
```

Gets the data this sample contains.

9.387.3.4 info() [2/2]

```
template<typename T >
const  WriteParams_t &  connext::WriteSample< T >::info ( )
```

Gets the write parameters.

The parameters are input and output to write operations. They can be set to configure the write operation (e.g. setting a specific publication timestamp) or they can be looked up afterward (e.g. what was the actual timestamp the sample was written with).

See also

connext::Requester::send_request(WriteSample<URep>&, const SampleIdentity_t&) (p. 2145)

9.387.3.5 identity()

```
template<typename T >
SampleIdentity_t  connext::WriteSample< T >::identity ( ) const  [inline]
```

Gets the identity of this sample.

The identity is assigned by the middleware after writing.

See also

Correlating requests and replies (p. 423)

References **DDS_WriteParams_t::identity**, and **connext::WriteSample< T >::info()**.

9.388 connext::WriteSampleRef< T > Class Template Reference

A reference to a data sample for writing.

Public Member Functions

- **WriteSampleRef** ()
  Creates an object with no data.
- **WriteSampleRef** (T & **data**, **WriteParams_t** &wparams)
  Creates an object with a reference to a data object.
- **WriteSampleRef** (**WriteSample**< T > &ws)
  Copies the references to data and params.
- T & **data** () const
  Gets the data this sample contains.
- **WriteParams_t** & **info** () const
  Gets the write parameters.
- void **set_data** (T & **data**)
  Sets a reference to the data to be written.
- void **set_info** (**WriteParams_t** & **info**)
  Copies the new parameters, overwriting the existing ones
- bool **is_nil_data** () const
  *Indicates if this **WriteSampleRef** (p. 2223) contains a reference to data or not.*
- **SampleIdentity_t** **identity** () const
  Gets the identity of this sample.

9.388.1 Detailed Description

template<typename T>
class connext::WriteSampleRef< T >

A reference to a data sample for writing.

A **WriteSampleRef** (p. 2223) can be used much like a **connext::WriteSample** (p. 2219), but it's different in that it holds references to data and write parameters, which need to be explicitly set to existing objects, whereas **WriteSample** (p. 2219) is a value type and it owns its data and info.

Template Parameters

T	The data type that this sample can contain

9.388.2 Constructor & Destructor Documentation

9.388.2.1 WriteSampleRef() [1/3]

```
template<typename T >
connext::WriteSampleRef< T >::WriteSampleRef ( )  [inline]
```

Creates an object with no data.

See also

set_data (p. 2225)

9.388.2.2 WriteSampleRef() [2/3]

```
template<typename T >
connext::WriteSampleRef< T >::WriteSampleRef (
            T & data,
             WriteParams_t & wparams )  [inline]
```

Creates an object with a reference to a data object.

See also

set_data (p. 2225)

9.388.2.3 WriteSampleRef() [3/3]

```
template<typename T >
connext::WriteSampleRef< T >::WriteSampleRef (
              WriteSample< T > & ws )  [inline]
```

Copies the references to data and params.

After this operation, this object contains a reference to the same data and WriteParams instances as the source.

9.388.3 Member Function Documentation

9.388.3.1 data()

```
template<typename T >
T &  connext::WriteSampleRef< T >::data ( ) const  [inline]
```

Gets the data this sample contains.

Precondition

Data must have been set before

See also

set_data (p. 2225)
connext::WriteSample::data() (p. 2221)

Referenced by **connext::WriteSampleRef< T >::set_data()**.

9.388.3.2 info()

```
template<typename T >
WriteParams_t &  connext::WriteSampleRef< T >::info ( ) const  [inline]
```

Gets the write parameters.

Precondition

The info object must have been set

See also

set_info() (p. 2226)
connext::WriteSample::identity() (p. 2222)

Referenced by **connext::WriteSampleRef< T >::identity()**, and **connext::WriteSampleRef< T >::set_info()**.

9.388.3.3 set_data()

```
template<typename T >
void  connext::WriteSampleRef< T >::set_data (
            T & data )  [inline]
```

Sets a reference to the data to be written.

Assigns a reference to the

Parameters

data	The data this sample will hold a reference to

References **connext::WriteSampleRef< T >::data()**.

9.388.3.4 set_info()

```
template<typename T >
void  connext::WriteSampleRef< T >::set_info (
              WriteParams_t & info )  [inline]
```

Copies the new parameters, overwriting the existing ones

References **connext::WriteSampleRef< T >::info()**.

9.388.3.5 is_nil_data()

```
template<typename T >
bool  connext::WriteSampleRef< T >::is_nil_data ( ) const  [inline]
```

Indicates if this **WriteSampleRef** (p. 2223) contains a reference to data or not.

Returns

True if this sample does not contain a reference to data.

See also

data() (p. 2225)
set_data (p. 2225)

9.388.3.6 identity()

```
template<typename T >
SampleIdentity_t  connext::WriteSampleRef< T >::identity ( ) const  [inline]
```

Gets the identity of this sample.

Precondition

The info object must have been set

See also

set_info() (p. 2226)
connext::WriteSample::identity() (p. 2222)

References **DDS_WriteParams_t::identity**, and **connext::WriteSampleRef< T >::info()**.

Chapter 10

File Documentation

10.1 howto_prototypes.h

```
1 /* $Id$
2
3  (c) Copyright, Real-Time Innovations, 2003-2016.
4  All rights reserved.
5  No duplications, whole or partial, manual or electronic, may be made
6  without express written permission.  Any such copies, or
7  revisions thereof, must display this notice unaltered.
8  This code contains trade secrets of Real-Time Innovations, Inc.
9
10  PUPROSE:
11    Participant use cases
12
13 modification history
14 --------------------
15 1.0h,24jul06,kaj replace long with DDS_Long
16 1.0d,02jun04,rw  Bug #8050: changed prototype of DataWriter::write()
17 1.0d,19apr04,eys Read/take parameters changed
18 1.0a,27oct03,gpc Written by combining pre-existing headers
19 ========================================================================== */
20
21 class FooDataWriter;
22 class Foo;
23
24 extern DDSDomainParticipantFactory* howto_get_factory();
25
26 extern DDSDomainParticipant*
27                howto_setup_participant(DDSDomainParticipantFactory* factory,
28                                        DDS_DomainId_t domain_id);
29
30 extern void howto_teardown_participant(DDSDomainParticipantFactory* factory,
31                                        DDSDomainParticipant* participant);
32
33 extern void howto_register_type(DDSDomainParticipant* participant,
34                                 const char* type_name);
35
36 extern DDSTopic* howto_setup_topic(DDSDomainParticipant* participant,
37                                    const char* topic_name, const char* type_name);
38
39 extern void howto_teardown_topic(DDSDomainParticipant* participant,
40                                  DDSTopic* topic);
41
42 extern DDSDataWriter* howto_setup_writer(DDSPublisher* publisher,
43                                          DDSTopic* topic);
44
45 extern void howto_send_data(FooDataWriter* writer, Foo& data,
46                             DDS_InstanceHandle_t instance_handle);
47
48 extern void howto_teardown_writer(DDSPublisher* publisher,
49                                   DDSDataWriter* writer);
50
51 extern DDSDataReader* howto_setup_reader(DDSSubscriber* subscriber,
```

```
52                                   DDSTopic* topic);
53
54 extern void howto_teardown_reader(DDSSubscriber* subscriber,
55                             DDSDataReader* reader);
56
57 extern void howto_setup_reader_to_access_received_data_via_listener(
58     DDSDataReader* entity, DDSDataReaderListener* entity_listener);
59
60 extern void howto_access_received_data_via_reader(
61     DDSDataReader* datareader,
62     DDS_Long max_samples,
63     DDS_SampleStateMask sample_state_mask,
64     DDS_ViewStateMask view_state_mask,
65     DDS_InstanceStateMask instance_state_mask);
66
67 extern DDSSubscriber* howto_setup_subscriber(DDSDomainParticipant* participant);
68
69 extern void howto_teardown_subscriber(DDSDomainParticipant* participant,
70                                 DDSSubscriber* subscriber);
71
72 extern void howto_setup_subscriber_to_access_received_data_via_listener(
73     DDSSubscriber* entity,  DDSSubscriberListener* entity_listener);
74
75 extern void howto_access_received_data_via_subscriber(DDSSubscriber* subscriber);
76
77 extern DDSPublisher* howto_setup_publisher(DDSDomainParticipant* participant);
78
79 extern void howto_teardown_publisher(DDSDomainParticipant* participant,
80                                 DDSPublisher* publisher);
```

10.2 transport_common.h

```
1 /*
2  * (c) Copyright, Real-Time Innovations, 2005-2024.
3  * All rights reserved.
4  *
5  * No duplications, whole or partial, manual or electronic, may be made
6  * without express written permission.  Any such copies, or
7  * revisions thereof, must display this notice unaltered.
8  * This code contains trade secrets of Real-Time Innovations, Inc.
9  */
10
11 #ifndef transport_common_h
12 #define transport_common_h
13
14 #include "osapi/osapi_type.h"
15 #include "osapi/osapi_rtpsGuid.h"
16 #include "reda/reda_buffer.h"
17 #include "transport/transport_dll.h"
18
19 /* Include the APIs useful for the transport plugin user */
20 #include "transport/transport_common_user.h"
21
22 #ifdef __cplusplus
23 extern "C" {
24 #endif
25
26 /* ================================================================= */
27 /*                     Fundamental types                             */
28 /* ================================================================= */
29 /*i
30  * \file
31  * \ingroup NDDS_TransportCommonComponent
32  * \brief Basic types and macros used by the \ndds Transport Plugin interface.
33  */
34
35 /* Data submessage ID */
36 #define NDDS_TRANSPORT_UDP_DATA_SUBMESSAGE_ID (0x15)
37
38 /* OID for SPDP participant announcement writer */
39 #define NDDS_TRANSPORT_RTPS_OBJECT_ID_SPDP_PARTICIPANT_WRITER (0x000100C2)
40
41 /* OID for SPDP2 bootstrap participant announcement writer */
42 #define NDDS_TRANSPORT_RTPS_OBJECT_ID_SPDP2_PARTICIPANT_BOOTSTRAP_WRITER \
43   (0x00010082)
44
45 /* OID for SPDP2 config participant writer */
```

```
46 #define NDDS_TRANSPORT_RTPS_OBJECT_ID_SPDP2_PARTICIPANT_CONFIG_WRITER \
47   (0x00010182)
48
49 #define NDDS_TRANSPORT_RTPS_OBJECT_ID_SPDP2_PARTICIPANT_SECURE_CONFIG_WRITER \
50   (0xFF010182)
51
52 /* OID for publications writer */
53 #define NDDS_TRANSPORT_RTPS_OBJECT_ID_PUB_WRITER (0x000003C2)
54
55 /* OID for subscriptions writer */
56 #define NDDS_TRANSPORT_RTPS_OBJECT_ID_SUB_WRITER (0x000004C2)
57
58 /* OID for publications reader */
59 #define NDDS_TRANSPORT_RTPS_OBJECT_ID_PUB_READER (0x000003C7)
60
61 /* OID for subscriptions reader */
62 #define NDDS_TRANSPORT_RTPS_OBJECT_ID_SUB_READER (0x000004C7)
63
64 /* OID for liveliness writer */
65 #define NDDS_TRANSPORT_RTPS_OBJECT_ID_P2P_PARTICIPANT_LIVELINESS_WRITER \
66   (0x000200C2)
67
68 /* OID for secure liveliness writer */
69 #define NDDS_TRANSPORT_RTPS_OBJECT_ID_SECURE_P2P_PARTICIPANT_LIVELINESS_WRITER \
70   (0xFF0200C2)
71
72 /* Auto liveliness kind */
73 #define NDDS_TRANSPORT_INTER_PARTICIPANT_MESSAGE_KIND_AUTOMATIC_LIVELINESS_UPDATE \
74   (0x00000001)
75
76 /* Manual livelienss kind */
77 #define NDDS_TRANSPORT_INTER_PARTICIPANT_MESSAGE_KIND_MANUAL_LIVELINESS_UPDATE \
78   (0x00000002)
79
80 /* ================================================================= */
81 /*                         Port Type                                 */
82 /* ================================================================= */
83
84 /*e
85  * \ingroup NDDS_TransportConfigurationComponent
86  *
87  * \brief Type for storing \ndds RTPS ports.
88  *
89  * Unlike IPv4 Socket API ports, which are 2 bytes long, the \ndds
90  * representation of an RTPS port is 4 bytes.
91  */
92 typedef RTI_UINT32 NDDS_Transport_Port_t;
93
94 /*e
95  * \ingroup NDDS_TransportConfigurationComponent
96  * \brief Port 0 is considered to be invalid.
97  */
98 #define NDDS_TRANSPORT_PORT_INVALID ((NDDS_Transport_Port_t) 0)
99
100 /*i
101  * \ingroup NDDS_TransportConfigurationComponent
102  * Represents a port that is shared across multiple users.
103  */
104 struct NDDS_Transport_SharedPort_t {
105     NDDS_Transport_Port_t port;
106     int ref_count;
107 };
108
109 /* ================================================================= */
110 /*                          UUID Type                                */
111 /* ================================================================= */
112
113 /*e
114  * \ingroup NDDS_TransportConfigurationComponent
115  * \brief Size of a NDDS_Transport_UUID.
116  */
117 #define NDDS_TRANSPORT_UUID_SIZE 12
118
119 /*e
120  * \ingroup NDDS_TransportConfigurationComponent
121  * \brief Represent an unlimited length
122  */
123 #define NDDS_TRANSPORT_LENGTH_UNLIMITED -1
124
125 /*e
126  * \ingroup NDDS_TransportConfigurationComponent
```

```
127  * \brief Rank interface as unknown or not yet set
128  */
129 #define NDDS_TRANSPORT_INTERFACE_RANK_UNKNOWN 0
130
131 /*e
132  * \ingroup NDDS_TransportConfigurationComponent
133  *
134  * \brief Unequivocally identifies a transport plugin instance.
135  */
136 struct NDDS_Transport_UUID {
137     unsigned char value[NDDS_TRANSPORT_UUID_SIZE];
138 };
139
140 /*e
141  * \ingroup NDDS_TransportConfigurationComponent
142  * \brief Value for UUIDs that have no known value. Used as default.
143  */
144 #define NDDS_TRANSPORT_UUID_UNKNOWN \
145   { \
146     { \
147       0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0 \
148     } \
149   }
150
151 /* ================================================================= */
152 /*                 (Network) Interface Type                          */
153 /* ================================================================= */
154 /*e
155  * \defgroup NDDS_Transport_Interface_t Interface
156  * \ingroup NDDS_TransportCommonComponent
157  * \brief Abstraction of a Transport Plugin network interface.
158  *
159  * A Transport Plugin may be able to use several logical or physical
160  * network interfaces in a single node (machine).  For example, there
161  * may be multiple NICs for IP networks or multiple serial ports for
162  * serial networks.
163  *
164  * An instance of a Transport Plugin is a conduit to one or more network
165  * interfaces associated with the transport.
166  *
167  * Instances of a Transport Plugin must assign a unique unicast
168  * address to each of the network interfaces that it can use to send
169  * and receive messages.  The unicast address should be within the
170  * range of addresses that are addressable by the Transport Plugin (as
171  * defined by the plugin itself).
172  *
173  * Then, when \ndds sends a message to an unicast destination address,
174  * the destination address will be made of two parts.  The network
175  * address and an interface address.  The network address portion will
176  * be used by \ndds to select the Transport Plugin instance that will
177  * send the message.  The interface address portion is passed to the
178  * Transport Plugin instance as the destination interface to which the
179  * message should be sent.
180  *
181  * @see NDDS_Transport_Address_t NDDS_Transport_ClassId_t
182  */
183
184 /*e
185   \ingroup NDDS_Transport_Interface_t
186   \brief Interface status
187  */
188 typedef enum {
189     /*e
190      * \brief The transport interface is OFF.
191      */
192     NDDS_TRANSPORT_INTERFACE_OFF = 0,
193     /*e
194      * \brief The transport interface is ON.
195      */
196     NDDS_TRANSPORT_INTERFACE_ON = 1
197 } NDDS_Transport_Interface_Status_t;
198
199 /*e
200  * \ingroup NDDS_Transport_Interface_t
201  * \brief Storage for the description of a network interface used by a
202  * Transport Plugin.
203  */
204 typedef struct {
205     /*e
206      * \brief The transport classid of the interface.
207      */
```

```
208     NDDS_Transport_ClassId_t transport_classid;
209
210     /*e
211      * \brief An unicast address that uniquely identifies this
212      * interface in the network specified by the transport class.
213      */
214     NDDS_Transport_Address_t address;
215
216     /*e
217      * \brief The state of the interface
218      */
219     NDDS_Transport_Interface_Status_t status;
220
221     /*e
222      * \brief Rank of the interface. Used when allow_interfaces_list Qos is set.
223      * A rank value will be assigned to each of the interfaces that match the
224      * allow_interfaces_list filter allowing an endpoint to prioritize some
225      * interfaces upon others.
226      */
227     RTI_UINT16 rank;
228
229 } NDDS_Transport_Interface_t;
230
231 #define NDDS_TRANSPORT_INTERFACE_INITIALIZER \
232   { \
233     NDDS_TRANSPORT_CLASSID_INVALID, \
234             NDDS_TRANSPORT_ADDRESS_INVALID_INITIALIZER, \
235             NDDS_TRANSPORT_INTERFACE_OFF, \
236             NDDS_TRANSPORT_INTERFACE_RANK_UNKNOWN \
237   }
238
239 /*i
240  * \ingroup NDDS_Transport_Interface_t
241  * \brief Prints the contents of an interface to standard out.
242  *
243  * @param interface_in \st_in Interface to be printed.
244  * @param desc_in    \st_in A prefix to be printed before the interface.
245  * @param indent_in  \st_in Indentation level for the printout.
246  */
247 extern NDDS_Transport_DllExport void NDDS_Transport_Interface_print(
248         const NDDS_Transport_Interface_t *interface_in,
249         const char *desc_in,
250         RTI_INT32 indent_in);
251
252 /*i
253  * \ingroup NDDS_Transport_Interface_t
254  * \brief Copy an interface from the source to the destination.
255  *
256  * @param dst_out \st_out From where the Interface is copied.
257  * @param src_in  \st_in  To where the Interface is copied.
258  */
259 extern NDDS_Transport_DllExport void NDDS_Transport_Interface_copy(
260         NDDS_Transport_Interface_t *dst_out,
261         const NDDS_Transport_Interface_t *src_in);
262
263 #define NDDS_TRANSPORT_CONTEXT_INVALID \
264   { \
265     NDDS_TRANSPORT_ADDRESS_INVALID_INITIALIZER, \
266     NDDS_TRANSPORT_ADDRESS_INVALID_INITIALIZER, \
267     NDDS_TRANSPORT_PORT_INVALID, \
268     NDDS_TRANSPORT_PORT_INVALID, \
269     NDDS_TRANSPORT_PORT_INVALID, \
270     NDDS_TRANSPORT_CLASSID_INVALID \
271   }
272
273 /*i
274  * \ingroup NDDS_TransportCommonComponent
275  * @brief Holds the transport information to be used by Network Capture.
276  *
277  * This struct is first filled by the transport (then associated to a
278  * RTINetioMessage if it is inbound traffic) and finally given as input to
279  * RTINetioCapManager_enqueueRtpsFrame.
280  */
281 struct NDDS_Transport_Context_t {
282     /*i
283      * @brief Source address of the message.
284      */
285     NDDS_Transport_Address_t sourceAddress;
286     /*i
287      * @brief Destination address of the message.
288      */
```

```
289     NDDS_Transport_Address_t destinationAddress;
290     /*i
291      * @brief source port of the message.
292      */
293     NDDS_Transport_Port_t sourcePort;
294     /*i
295      * @brief Destination port of the message.
296      * In the case of TCP, if participant is acting as server,
297      * this is the server bind port.
298      */
299     NDDS_Transport_Port_t destinationPort;
300     /*i
301      * @brief RTPS port when TCP and participant acting as server.
302      * Otherwise, destinationRTPSPort is equal to 0 and we are assuming
303      * destinationRTPSPort==destinationPort.
304      */
305     NDDS_Transport_Port_t destinationRTPSPort;
306     /*i
307      * @brief Transport class Id. This information is filled by netio.
308      */
309     NDDS_Transport_ClassId_t classId;
310 };
311
312 /* ================================================================= */
313 /*                         Transport Priority                        */
314 /* ================================================================= */
315 /*i
316  * \defgroup NDDS_TransportPriority Transport Priority
317  * \ingroup NDDS_TransportCommonComponent
318  *
319  * \brief A QoS set in DDS DataWriters, \idref_TransportPriorityQosPolicy
320  * that is passed to the Transport Plugin when creating SendResources.
321  *
322  * Sometimes also referred to as Transport QoS or CoS (class of
323  * service), a transport priority at the Transport Plugin level is
324  * directly set by the value of the \idref_TransportPriorityQosPolicy
325  * set on DataWriters. The \ndds core will pass the value of the
326  * \idref_TransportPriorityQosPolicy to the \c share/create SendResource
327  * functions of the Transport API when a DataWriter is created.
328  *
329  * The \ndds core itself does not do anything with the transport priority
330  * except passing it from the user code that set the value via the DDS
331  * API to the Transport API.
332  *
333  * Only Transport Plugin implementations that have the concept of
334  * sending of messages at different priorities or with different
335  * classes of service (CoSs) will find this support useful.  It is up
336  * to each implementation to use the transport priority in what ever
337  * way it can to configure itself.  Transport Plugins implementations
338  * that do not have a concept of sending data with different priorities
339  * can ignore it.
340  *
341  * For example, TCP/IP-based plugins may offer the user application the
342  * ability to specify different CoS for DataWriters so that individual
343  * message streams may be treated differently by the physical transport
344  * itself.
345  */
346
347 /*i
348  * \ingroup NDDS_TransportPriority
349  * \brief The default value of the transport_priority.
350  */
351 #define NDDS_TRANSPORT_PRIORITY_DEFAULT (0)
352
353 /* ================================================================= */
354 /*                           Buffer Type                             */
355 /* ================================================================= */
356 /*i \ingroup NDDS_TransportCommonComponent
357  * \brief Generic buffer to hold data, described by a pointer to the
358  * data and length of the data.
359  */
360 #ifdef DOXYGEN_DOCUMENTATION_ONLY
361 typedef struct {
362     /*i @brief Length of the buffer pointed to by pointer. */
363     RTI_INT32 length;
364     /*i @brief Pre-allocated (by the caller) buffer. */
365     char *pointer;
366 } NDDS_Transport_Buffer_t;
367 #endif
368 typedef struct REDABuffer NDDS_Transport_Buffer_t;
369
```

```
370 /*
371  * Structure subject to hidden padding in 64bit build,
372  * so use field compare instead of memory compare.
373  */
374 #define NDDS_Transport_Buffer_t_compare(transportBuffer1, transportBuffer2) \
375   (((transportBuffer1)->length < (transportBuffer2)->length) \
376           ? -1 \
377           : (((transportBuffer1)->length > (transportBuffer2)->length) \
378                   ? 1 \
379                   : (((transportBuffer1)->pointer \
380                       < (transportBuffer2)->pointer) \
381                              ? -1 \
382                              : (((transportBuffer1)->pointer \
383                                  > (transportBuffer2)->pointer) \
384                                         ? 1 \
385                                         : 0))))
386
393 extern NDDS_Transport_DllExport RTIBool NDDS_Transport_get_address(
394         char *addresses_str_out,
395         size_t addresses_str_buffer_size_in,
396         const char *list,
397         RTI_UINT32 index);
398
405 extern NDDS_Transport_DllExport RTIBool
406 NDDS_Transport_get_number_of_addresses_in_string(
407         RTI_UINT32 *number_out,
408         const char *list);
409
410 #ifdef __cplusplus
411 } /* extern "C" */
412 #endif
413
414 #include "transport/transport_common_impl.h"
415
416 #endif /* transport_common_h */
```

10.3 transport_common_user.h

```
1 /*
2  * (c) Copyright, Real-Time Innovations, 2005-2024.
3  * All rights reserved.
4  *
5  * No duplications, whole or partial, manual or electronic, may be made
6  * without express written permission.  Any such copies, or
7  * revisions thereof, must display this notice unaltered.
8  * This code contains trade secrets of Real-Time Innovations, Inc.
9  */
10
11 #ifndef transport_common_user_h
12 #define transport_common_user_h
13
14 #include "osapi/osapi_type.h"
15 #include "osapi/osapi_socket.h"
16 #include "reda/reda_buffer.h"
17 #include "transport/transport_dll.h"
18
19 #ifdef __cplusplus
20 extern "C" {
21 #endif
22
23 /* ========================================================================= */
24 /*                       Fundamental types                                   */
25 /* ========================================================================= */
26 /*i
27  * \file
28  * \ingroup NDDS_TransportCommonComponent
29  * \brief Basic types and macros used by the \ndds Transport Plugin interface.
30  */
31
32 /*e \dref_TransportConfigurationGroupDocs
33  */
34
35 /* ========================================================================= */
36 /*                   Transport Allocation Settings                           */
37 /* ========================================================================= */
38 /*e \ingroup NDDS_TransportConfigurationComponent
39  *
```

```
40  * \brief Allocation settings used by various internal buffers.
41  *
42  * An allocation setting structure defines the rules of memory management
43  * used by internal buffers.
44  *
45  * An internal buffer can provide blocks of memory of fixed size. They
46  * are used in several places of any transport, and this structure
47  * defines its starting size, limits, and how to increase its capacity.
48  *
49  * It contains three values:
50  * <ul><li>initial_count: the number of individual elements that are
51  *         allocated when the buffer is created.
52  *
53  *     <li>max_count: the maximum number of elements the buffer can
54  *         hold. The buffer will grow up to this amount. After this
55  *         limit is reached, new allocation requests will fail.
56  *         For unlimited size, use the value
57  *         NDDS_TRANSPORT_ALLOCATION_SETTINGS_MAX_COUNT_UNLIMITED
58  *
59  *     <li>incremental_count: The amount of elements that are allocated
60  *         at every increment.
61  *         You can use the value:
62  *         NDDS_TRANSPORT_ALLOCATION_SETTINGS_INCREMENTAL_COUNT_AUTOMATIC
63  *         to have the buffer double its size at every reallocation request.
64  * </ul>
65  */
66 struct TransportAllocationSettings_t {
67     RTI_INT32 initial_count;
68     RTI_INT32 max_count;
69     RTI_INT32 incremental_count;
70 };
71
72 /*e \ingroup NDDS_TransportConfigurationComponent
73  *
74  * \brief The constant used as 'unlimited' for the 'max_count' field of the
75  *        structure \ref TransportAllocationSettings_t
76  */
77 #define NDDS_TRANSPORT_ALLOCATION_SETTINGS_MAX_COUNT_UNLIMITED          (-1)
78
79 /*e \ingroup NDDS_TransportConfigurationComponent
80  *
81  * \brief The constant used as 'automatic' for the 'incremental_count' field of
82  * the structure \ref TransportAllocationSettings_t
83  *
84  * Automatic means the buffer size will double at every reallocation.
85  */
86 #define NDDS_TRANSPORT_ALLOCATION_SETTINGS_INCREMENTAL_COUNT_AUTOMATIC  (-1)
87
88 /*e \ingroup NDDS_TransportConfigurationComponent
89  *
90  * \brief The constant used as default value for the struct \ref
91  * TransportAllocationSettings_t
92  *
93  * The default value defined in this constant, sets the buffer to have:
94  * <ul><li>initial_count = 2 elements
95  *     <li>max_count = \ref
96  * NDDS_TRANSPORT_ALLOCATION_SETTINGS_MAX_COUNT_UNLIMITED <li>incremental_count
97  * = NDDS_TRANSPORT_ALLOCATION_SETTINGS_INCREMENTAL_COUNT_AUTOMATIC
98  * </ul>
99  */
100 #define NDDS_TRANSPORT_ALLOCATION_SETTINGS_DEFAULT {                              \
101         2L, /* initial_count */                                                   \
102         NDDS_TRANSPORT_ALLOCATION_SETTINGS_MAX_COUNT_UNLIMITED, /* max_count */     \
103         NDDS_TRANSPORT_ALLOCATION_SETTINGS_INCREMENTAL_COUNT_AUTOMATIC /* incremental_count */ \
104 }
105
106 /* ================================================================= */
107 /*                  Transport Class ID type                          */
108 /* ================================================================= */
109
110 /*e
111  * \ingroup NDDS_TransportConfigurationComponent
112  * \brief Type for storing \ndds Transport Plugin class IDs.
113  *
114  * Each implementation of a Transport Plugin must have a unique ID.  For
115  * example, a UDP/IP Transport Plugin implementation must have a different
116  * ID than a Shared Memory Transport Plugin.
117  *
118  * User-implemented Transport Plugins must have an ID higher than \ref
119  * NDDS_TRANSPORT_CLASSID_RESERVED_RANGE.
120  */
```

```
121 typedef RTI_INT32 NDDS_Transport_ClassId_t;
122
123 #define NDDS_TRANSPORT_CLASS_ID_LIST_SIZE 8
124
125 /*i
126  * \ingroup NDDS_TransportConfigurationComponent
127  * @brief Array of NDDS_Transport_ClassId_t elements
128  *
129  * Used in Network Capture as a list of transports to capture frames from.
130  * This will be populated by
131  * RTINetioConfigurator_getTransportClassIdListFromString
132  * with the list of transports (given by the user)
133  * and then checked against the transport of the current frame in
134  * RTINetioCapManager_enqueueRtpsFrame
135  */
136 struct NDDS_Transport_ClassId_List {
137     size_t size;
138     NDDS_Transport_ClassId_t elements[NDDS_TRANSPORT_CLASS_ID_LIST_SIZE];
139 };
140
141 /*e
142  * \ingroup NDDS_TransportConfigurationComponent
143  * \brief Invalid Transport Class ID
144  *
145  * Transport-Plugins implementations should set their class ID to a value
146  * different than this.
147  */
148 #define NDDS_TRANSPORT_CLASSID_INVALID (-1)
149
150
151 #define NDDS_Transport_ClassId_List_INITIALIZER \
152 { \
153     0, /* size */ \
154     { \
155         NDDS_TRANSPORT_CLASSID_INVALID, \
156         NDDS_TRANSPORT_CLASSID_INVALID, \
157         NDDS_TRANSPORT_CLASSID_INVALID, \
158         NDDS_TRANSPORT_CLASSID_INVALID, \
159         NDDS_TRANSPORT_CLASSID_INVALID, \
160         NDDS_TRANSPORT_CLASSID_INVALID, \
161         NDDS_TRANSPORT_CLASSID_INVALID, \
162         NDDS_TRANSPORT_CLASSID_INVALID \
163     } \
164 }
165
166 /*i \ingroup NDDS_TransportConfigurationComponent
167  * \brief Special \b wildcard transport class. Matches any transport class. A
168  * locator with this transport class is applicable to ALL Transport Plugin
169  * instances.
170  *
171  * Physical Transport-Plugins should return a value different than this.
172  *
173  * Used by \ndds internally, not useful for Transport Plugin implementors.
174  */
175 #define NDDS_TRANSPORT_CLASSID_ANY (0)
176
177 /*e
178  * \ingroup NDDS_TransportConfigurationComponent
179  * \brief Builtin IPv4 UDP/IP Transport Plugin class ID.
180  */
181 #define NDDS_TRANSPORT_CLASSID_UDPv4 (1)
182
183 /*e
184  * \ingroup NDDS_TransportConfigurationComponent
185  * \brief Builtin Shared-Memory Transport Plugin class ID.
186  */
187 #define NDDS_TRANSPORT_CLASSID_SHMEM (0x01000000)
188
189 /*e
190  * \ingroup NDDS_TransportConfigurationComponent
191  * \brief Builtin Shared-Memory Transport Plugin class ID for Connext 5.1.0 and
192  * earlier.
193  */
194 #define NDDS_TRANSPORT_CLASSID_SHMEM_510 (2)
195
196 /*i
197  * \ingroup NDDS_TransportConfigurationComponent
198  * \brief Builtin Intra-Process Transport Plugin class ID.
199  */
200 #define NDDS_TRANSPORT_CLASSID_INTRA (3)
201
```

```
202 /* NOTE: classid 4 was previously used for starfabric */
203
204 /*e
205  * \ingroup NDDS_TransportConfigurationComponent
206  * \brief Builtin IPv6 UDP/IP Transport Plugin class ID.
207  */
208 #define NDDS_TRANSPORT_CLASSID_UDPv6 (2)
209
210 /*e
211  * \ingroup NDDS_TransportConfigurationComponent
212  * \brief Builtin IPv6 UDP/IP Transport Plugin class ID for Connext 5.1.0 and
213  * earlier.
214  */
215 #define NDDS_TRANSPORT_CLASSID_UDPv6_510 (5)
216
217 /*e
218  * \ingroup NDDS_TransportConfigurationComponent
219  * \brief   IPv4 TCP/IP Transport Plugin class ID for LAN case
220  */
221 #define NDDS_TRANSPORT_CLASSID_TCPV4_LAN (8)
222
223 /*e
224  * \ingroup NDDS_TransportConfigurationComponent
225  * \brief IPv4 TCP/IP Transport Plugin class ID for WAN case
226  */
227 #define NDDS_TRANSPORT_CLASSID_TCPV4_WAN (9)
228
229 /*e
230  * \ingroup NDDS_TransportConfigurationComponent
231  * \brief IPv4 TCP/IP Transport Plugin class name for WAN case
232  */
233 #define NDDS_TRANSPORT_CLASSNAME_TCPV4_WAN "tcpv4_wan"
234
235 /*e
236  * \ingroup NDDS_TransportConfigurationComponent
237  * \brief IPv4 TCP/IP Transport Plugin class ID for LAN case with TLS enabled
238  */
239 #define NDDS_TRANSPORT_CLASSID_TLSV4_LAN (10)
240
241 /*e
242  * \ingroup NDDS_TransportConfigurationComponent
243  * \brief   IPv4 TCP/IP Transport Plugin class ID for WAN case with TLS enabled
244  */
245 #define NDDS_TRANSPORT_CLASSID_TLSV4_WAN (11)
246
247 /*i
248  * \ingroup NDDS_TransportConfigurationComponent
249  * \brief PCIE Transport Plugin class ID.
250  */
251 #define NDDS_TRANSPORT_CLASSID_PCIE (12)
252
253 /*i
254  * \ingroup NDDS_TransportConfigurationComponent
255  * \brief Internet Transport Plugin class ID.
256  */
257 #define NDDS_TRANSPORT_CLASSID_ITP (13)
258
259 /*e
260  * \ingroup NDDS_TransportConfigurationComponent
261  * \brief Builtin IPv4 UDP/IP Asymmetric Transport Plugin class ID.
262  */
263 #define NDDS_TRANSPORT_CLASSID_UDPv4_WAN (0x01000001)
264
265 /*e
266  * \ingroup NDDS_TransportConfigurationComponent
267  * \brief Transport Plugin class IDs below this are reserved by RTI.
268  *
269  * User-defined Transport-Plugins should use a class ID greater than
270  * this number.
271  */
272 #define NDDS_TRANSPORT_CLASSID_RESERVED_RANGE (1000)
273
274 #define NDDS_Transport_ClassId_ignore_participant_index_for_initial_peers( \
275         class_id__) \
276     (class_id__ == NDDS_TRANSPORT_CLASSID_UDPv4_WAN)
277
278 /* Class name definitions for pluggable transports */
279 #define NDDS_TRANSPORT_IP_CLASSNAME_TCPV4_LAN "tcpv4_lan"
280 #define NDDS_TRANSPORT_IP_CLASSNAME_TCPV4_WAN "tcpv4_wan"
281 #define NDDS_TRANSPORT_IP_CLASSNAME_TLSV4_LAN "tlsv4_lan"
282 #define NDDS_TRANSPORT_IP_CLASSNAME_TLSV4_WAN "tlsv4_wan"
```

```
283
284 /* ================================================================= */
285 /*                  Transport Short Name                             */
286 /* ================================================================= */
287 /*i \ingroup NDDSTransportModule
288  * \brief Transport short name.  Specifies the short name of the transports.
289  */
290 #define NDDS_TRANSPORT_SHORTNAME_SHMEM "SHMEM"
291 #define NDDS_TRANSPORT_SHORTNAME_TCPV4 "TCP4"
292 #define NDDS_TRANSPORT_SHORTNAME_TLS "TLS"
293 #define NDDS_TRANSPORT_SHORTNAME_WAN "WAN"
294 #define NDDS_TRANSPORT_SHORTNAME_UDP4 "UDP4"
295 #define NDDS_TRANSPORT_SHORTNAME_UDP6 "UDP6"
296
297 /* ================================================================= */
298 /*                          Address Type                             */
299 /* ================================================================= */
300
301 /*e
302  * \defgroup NDDS_Transport_Address_t Transport Address
303  * \ingroup NDDSTransportModule
304  *
305  * \brief Transport-independent addressing scheme using IPv6
306  * presentation strings and numerically stored in network-ordered format.
307  *
308  * The APIs of \ndds uses IPv6 address notation for all transports.
309  *
310  * Transport Plugin implementations that are not IP-based are required to map
311  * whatever addressing scheme natively used by the physical transport (if any)
312  * to an address in IPv6 notation and vice versa.
313  *
314  * IPv6 addresses are numerically stored in 16 bytes. An IPv6 address can be
315  * presented In string notation in a variety of ways. For example,
316  *
317  * \code
318  * "00AF:0000:0037:FE01:0000:0000:034B:0089"
319  * "AF:0:37:FE01:0:0:34B:89"
320  * "AF:0:37:FE01::34B:89"
321  * \endcode
322  *
323  * are all valid IPv6 presentation of the same address.
324  *
325  * IPv4 address in dot notation can be used to specify the last 4 bytes of the
326  * address.  For example,
327  *
328  * \code
329  * "0000:0000:0000:0000:0000:0000:192.168.0.1"
330  * "0:0:0:0:0:0:192.168.0.1"
331  * "::192.168.0.1"
332  * \endcode
333  *
334  * are all valid IPv6 presentation of the same address.
335  *
336  * For a complete description of valid IPv6 address notation, consult the IPv6
337  * Addressing Architecture (RFC 2373).
338  *
339  * Addresses are divided into unicast addresses and multicast addresses.
340  *
341  * Multicast addresses are defined as
342  *
343  * \li Addresses that start with 0xFF.
344  * That is \c FFxx:xxxx:xxxx:xxxx:xxxx:xxxx:xxxx:xxxx.
345  *
346  * or an IPv4 multicast address
347  *
348  * \li Address in the range \c [::224.0.0.0, ::239.255.255.255]
349  *
350  * Multicast addresses do not refer to any specific destination (network
351  * interface). Instead, they usually refer to a group of network interfaces,
352  * often called a "multicast group".
353  *
354  * Unicast addresses always refer to a specific network interface.
355  *
356  */
357
358 #define NDDS_TRANSPORT_ADDRESS_LENGTH (16)
359 #define NDDS_TRANSPORT_ADDRESS_BIT_LENGTH (128)
360
361 /*e
362  * \ingroup NDDS_Transport_Address_t
363  * \brief Addresses are stored individually as network-ordered bytes.
```

```
364  *
365  * \ndds addresses are numerically stored in a transport independent manner.
366  * \ndds uses a IPv6-compatible format, which means that the data structure to
367  * hold an NDDS_Transport_Address_t is the same size as a data structure needed
368  * to hold an IPv6 address.
369  *
370  * In addition, the functions provided to translate a string representation of
371  * an \ndds address to a value assumes that the string presentation follows the
372  * IPv6 address presentation as specified in RFC 2373.
373  *
374  * An NDDS_Transport_Address_t always stores the address in network-byte order
375  * (which is Big Endian).
376  *
377  * For example, IPv4 multicast address of 225.0.0.0 is represented by
378  *
379  * {{0,0,0,0, 0,0,0,0, 0,0,0,0, 0xE1,0,0,0}} regardless of endianness,
380  *
381  * where 0xE1 is the 13th byte of the structure (\c network_ordered_value[12]).
382  *
383  */
384 typedef struct {
385     /*e
386      * network-byte ordered (i.e., bit 0 is the most significant bit and bit 128
387      * is the least significant bit).
388      */
389     unsigned char network_ordered_value[NDDS_TRANSPORT_ADDRESS_LENGTH];
390 } NDDS_Transport_Address_t;
391
392 /*i
393  * @brief Converts an IPv4 address to string.
394  *
395  * @return
396  * RTI_TRUE upon success; RTI_FALSE upon failure (not enough space in the
397  * provided buffer)
398  */
399 extern NDDS_Transport_DllExport RTIBool NDDS_Transport_v4Address_to_string(
400         const NDDS_Transport_Address_t *me,
401         char *buffer,
402         size_t buffer_size_in);
403
404 /*e
405  * \ingroup NDDS_Transport_Address_t
406  * \brief Converts a numerical address to a printable string representation.
407  *
408  * @pre The \c buffer_inout provided must be at least \ref
409  * NDDS_TRANSPORT_ADDRESS_STRING_BUFFER_SIZE characters long.
410  *
411  * @param self \st_in The address to be converted.
412  *
413  * @param buffer_inout \st_inout Storage passed in which to return the string
414  * corresponding to the address.
415  *
416  * @param buffer_size_in \st_in The size of the storage buffer. Must be
417  * >= \ref NDDS_TRANSPORT_ADDRESS_STRING_BUFFER_SIZE
418  *
419  * @return 1 upon success; 0 upon failure (not enough space in the provided
420  * buffer)
421  */
422 extern NDDS_Transport_DllExport RTIBool NDDS_Transport_Address_to_string(
423         const NDDS_Transport_Address_t *self,
424         char *buffer_inout,
425         size_t buffer_size_in);
426
427 /*e
428  * @brief Converts an address to string given its class ID.
429  *
430  * @return
431  * 1 upon success; 0 upon failure (not enough space in the provided buffer)
432  */
433 extern NDDS_Transport_DllExport RTIBool
434 NDDS_Transport_Address_to_string_with_class_id(
435         const NDDS_Transport_Address_t *me,
436         char *buffer,
437         size_t buffer_size_in,
438         NDDS_Transport_ClassId_t transport_class_id);
439
440 /*e
441  * \ingroup NDDS_Transport_Address_t
442  * \brief Converts a numerical address to a printable string representation with
443  * IPv4 dotted notation or IPv6 presentation string depending on the provided
444  * protocol family.
```

```
445  *
446  * @pre The \c buffer_inout provided must be at least
447  * \ref NDDS_TRANSPORT_ADDRESS_STRING_BUFFER_SIZE characters long.
448  *
449  * @param me \st_in The address to be converted.
450  *
451  * @param buffer \st_inout Storage passed in which to return the string
452  * corresponding to the address.
453  *
454  * @param buffer_size_in \st_in The size of the storage buffer. Must be
455  * >= \ref NDDS_TRANSPORT_ADDRESS_STRING_BUFFER_SIZE
456  *
457  * @param family \st_in The protocol family RTI_OSAPI_SOCKET_AF_INET or
458  * RTI_OSAPI_SOCKET_AF_INET6
459  *
460  * @return RTI_TRUE upon success; 0 upon failure (not enough space in the
461  * provided buffer)
462  */
463 extern NDDS_Transport_DllExport RTIBool
464 NDDS_Transport_Address_to_string_with_protocol_family_format(
465         const NDDS_Transport_Address_t *me,
466         char *buffer,
467         size_t buffer_size_in,
468         RTIOsapiSocketAFKind family);
469
470 /*e
471  * \ingroup NDDS_Transport_Address_t
472  * \brief Converts an address (IPv4 dotted notation or IPv6 presentation string)
473  * into a numerical address.
474  *
475  * The address string must be in IPv4 dotted notation (X.X.X.X) or IPv6
476  * presentation notation.  The string cannot be a hostname since this
477  * function does not perform a hostname lookup.
478  *
479  * @param address_out \st_out Numerical value of the address.
480  *
481  * @param address_in  \st_in  String representation of an address.
482  *
483  * @return 1 if address_out contains a valid address, 0 if it was not able to
484  * convert the string into an address.
485  */
486 extern NDDS_Transport_DllExport RTI_INT32 NDDS_Transport_Address_from_string(
487         NDDS_Transport_Address_t *address_out,
488         const char *address_in);
489
490 /*e
491  * \ingroup NDDS_Transport_Address_t
492  * \brief Prints an address to standard out.
493  *
494  * @param address_in \st_in Address to be printed.
495  * @param desc_in    \st_in A prefix to be printed before the address.
496  * @param indent_in  \st_in Indentation level for the printout.
497  */
498 extern NDDS_Transport_DllExport void NDDS_Transport_Address_print(
499         const NDDS_Transport_Address_t *address_in,
500         const char *desc_in,
501         RTI_INT32 indent_in);
502
503 /*i
504  * \ingroup NDDS_Transport_Address_t
505  * \brief Copies an address.
506  *
507  *  @param dst_out \st_out To where the address is copied.
508  *  @param src_in  \st_in  From where the address is copied.
509  */
510 extern NDDS_Transport_DllExport void NDDS_Transport_Address_copy(
511         NDDS_Transport_Address_t *dst_out,
512         const NDDS_Transport_Address_t *src_in);
513
514 /*i
515  * \ingroup NDDS_Transport_Address_t
516  * \brief Compares two addresses for equality.
517  *
518  * Does the comparison (l_in == r_in) ?
519  *
520  * @param l_in \st_in Address left.
521  * @param r_in \st_in Address right.
522  *
523  * @return 1 if addresses are equal, 0 otherwise.
524  */
525 extern NDDS_Transport_DllExport RTI_INT32 NDDS_Transport_Address_is_equal(
```

```
526         const NDDS_Transport_Address_t *l_in,
527         const NDDS_Transport_Address_t *r_in);
528
529 /*i
530  * \ingroup NDDS_Transport_Address_t
531  * \brief Compares two addresses for equality using only significant bits.
532  *
533  * Does the comparison (l_in == r_in) using only the number of
534  * significant bits as indicated by \c transport_address_bit_count_in.
535  *
536  * @param l_in \st_in Address left.
537  * @param r_in \st_in Address right.
538  * @param transport_address_bit_count_in \st_in Only pays attention to this
539  * many least significant (i.e. right-most) bits in the address. The most
540  * significant (i.e. lesftmost) \c 128 - \c address_bit_count_in bits are
541  * ignored.
542  *
543  * @return 1 if the \c address_bit_count_in least significant (i.e. rightmost)
544  * bits are equal,0 otherwise.
545  */
546 extern NDDS_Transport_DllExport RTI_INT32 NDDS_Transport_Address_bits_are_equal(
547         const NDDS_Transport_Address_t *l_in,
548         const NDDS_Transport_Address_t *r_in,
549         RTI_INT32 transport_address_bit_count_in);
550
551 /*i
552  * \ingroup NDDS_Transport_Address_t
553  * \brief Compares the value of two addresses.
554  *
555  * This function treats the addresses as 16-byte numerical values, and
556  * then compares them.
557  *
558  * @param l_in \st_in Address left.
559  * @param r_in \st_in Address right.
560  *
561  * @return 1 if l_in > r_in, 0 if l_ in == r_in, -1 if l_in < r
562  */
563 extern NDDS_Transport_DllExport RTI_INT32 NDDS_Transport_Address_compare(
564         const NDDS_Transport_Address_t *l_in,
565         const NDDS_Transport_Address_t *r_in);
566
567 /*e
568  * \ingroup NDDS_Transport_Address_t
569  * \brief Checks if an address is an IPv4 address.
570  *
571  * @param address_in \st_in Address to be tested.
572  *
573  * @note May be implemented as a macro for efficiency.
574  *
575  * @return 1 if address is an IPv4 address, 0 otherwise.
576  */
577 extern NDDS_Transport_DllExport RTI_INT32
578 NDDS_Transport_Address_is_ipv4(const NDDS_Transport_Address_t *address_in);
579
580 /*e
581  * \ingroup NDDS_Transport_Address_t
582  * \brief Checks if an address is an IPv4 or IPv6 multicast address.
583  *
584  * @param address_in \st_in Address to be tested.
585  *
586  * May be implemented as a macro for efficiency.
587  *
588  * @return 1 if address is a multicast address, 0 otherwise.
589  */
590 extern NDDS_Transport_DllExport RTI_INT32
591 NDDS_Transport_Address_is_multicast(const NDDS_Transport_Address_t *address_in);
592
593 /*i
594  * \ingroup NDDS_Transport_Address_t
595  * @brief Fill from host byte order IPv4 address
596  */
597 extern NDDS_Transport_DllExport void NDDS_Transport_Address_from_ipv4_host_byte(
598         NDDS_Transport_Address_t *me,
599         RTI_UINT32 ipv4AddressInHostOrderIn);
600
601 /*i
602  * \ingroup NDDS_Transport_Address_t
603  * @brief convert to host byte order IPv4 address
604  */
605 extern NDDS_Transport_DllExport RTI_UINT32
606 NDDS_Transport_Address_to_ipv4_host_byte(const NDDS_Transport_Address_t *me);
```

```
607
608 /*e
609  * \ingroup NDDS_Transport_Address_t
610  * \brief An invalid transport address. Used as an initializer.
611  *
612  *  For example:
613  *  NDDS_Transport_Address_t address =
614  *          NDDS_TRANSPORT_ADDRESS_INVALID_INITIALIZER;
615  */
616 #define NDDS_TRANSPORT_ADDRESS_INVALID_INITIALIZER \
617   { \
618     { \
619       0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0 \
620     } \
621   }
622
623 /*e
624  * \ingroup NDDS_Transport_Address_t
625  * \brief An invalid transport address.
626  */
627 extern NDDS_Transport_DllVariable const NDDS_Transport_Address_t
628         NDDS_TRANSPORT_ADDRESS_INVALID;
629
630 /*e
631  * \ingroup NDDS_Transport_Address_t
632  *  \brief The minimum size of the buffer that should be passed to
633  *  \ref NDDS_Transport_Address_to_string.
634  *
635  *  For regular addresses, the string size needs to be at least 40 to include
636  *  space for 8 tuples of 4 characters each plus 7 delimiting colons plus a
637  *  terminating NULL.
638  *
639  * To support UDPv4_WAN strings, it has been adjusted to 72 to fit the following
640  * representation (plus NULL terminator):
641  * f=XXXXRBPU,u={FF,FF,FF,FF,FF,FF,FF,FF,FF},p=255.255.255.255:65555:65555
642  */
643 #define NDDS_TRANSPORT_ADDRESS_STRING_BUFFER_SIZE (72)
644 #define NDDS_TRANSPORT_V4_ADDRESS_STRING_BUFFER_SIZE (16)
645
646 /* Functions that forward information from transport to dds_c */
647
648 /*
649  * Called when preallocating memory for encoded content.
650  *
651  * Unlike the participant forwarder APIs, the transport ones don't have the
652  * worker as an input argument. They get it from the DomainParticipant when
653  * they require it. They do so because the concept of worker is not currently
654  * part of the pluggable transports APIs.
655  */
656 typedef unsigned int (
657         *NDDS_Transport_Plugin_DomainParticipant_Get_Transformed_Outgoing_Message_Size_Function)(
658         void *domain_participant_ptr,
659         unsigned int message_size);
660
661 /*
662  * Unlike the participant forwarder APIs, the transport ones don't have the
663  * worker as an input argument. They get it from the DomainParticipant when
664  * they require it. They do so because the concept of worker is not currently
665  * part of the pluggable transports APIs.
666  */
667 typedef RTIBool (
668         *NDDS_Transport_Plugin_DomainParticipant_Transform_Outgoing_Message_Function)(
669         void *domain_participant_ptr,
670         /* output */
671         struct REDABuffer *transformed_message,
672         /* inputs */
673         const struct REDABuffer *message_buffers,
674         int message_buffer_count);
675
676 /*
677  * Unlike the participant forwarder APIs, the transport ones don't have the
678  * worker as an input argument. They get it from the DomainParticipant when
679  * they require it. They do so because the concept of worker is not currently
680  * part of the pluggable transports APIs.
681  */
682 typedef RTIBool (
683         *NDDS_Transport_Plugin_DomainParticipant_Transform_Incoming_Message_Function)(
684         void *domain_participant_ptr,
685         /* output */
686         struct REDABuffer *transformed_message,
687         /* input */
```

```
688         const struct REDABuffer *message_buffer);
689
690 struct NDDS_Transport_Plugin_DomainParticipantForwarder {
691     NDDS_Transport_Plugin_DomainParticipant_Get_Transformed_Outgoing_Message_Size_Function
692             getTransformedOutgoingMessageSize;
693     NDDS_Transport_Plugin_DomainParticipant_Transform_Outgoing_Message_Function
694             transformOutgoingMessage;
695     NDDS_Transport_Plugin_DomainParticipant_Transform_Incoming_Message_Function
696             transformIncomingMessage;
697 };
698
699 #ifdef __cplusplus
700 } /* extern "C" */
701 #endif
702
703 #endif /* transport_common_user_h */
```

10.4 transport_interface_user.h

```
1 /*
2  * (c) Copyright, Real-Time Innovations, 2005-2024.
3  * All rights reserved.
4  *
5  * No duplications, whole or partial, manual or electronic, may be made
6  * without express written permission.  Any such copies, or
7  * revisions thereof, must display this notice unaltered.
8  * This code contains trade secrets of Real-Time Innovations, Inc.
9  */
10
11 #ifndef transport_interface_user_h
12 #define transport_interface_user_h
13
14 #include "transport/transport_dll.h"
15 #include "transport/transport_common.h"
16
17 #ifdef __cplusplus
18 extern "C" {
19 #endif
20
21 /* ================================================================= */
22 /*       Component: Custom Transport Plugin (Abstract)               */
23 /* ================================================================= */
24
25 /*i \file
26  * \ingroup NDDS_TransportPluginComponent
27  * \brief Abstract transport Plugin API visible to the transport plugin user.
28  */
29
30
31 /* ================================================================= */
32 /*                          Transport Property                       */
33 /* ================================================================= */
34
35 /*e \dref_Transport_Property_t
36  * \ingroup NDDS_TransportConfigurationComponent
37  */
38 struct NDDS_Transport_Property_t {
39     /*e \dref_Transport_Property_t_classid */
40     NDDS_Transport_ClassId_t classid;
41
42     /*e \dref_Transport_Property_t_address_bit_count */
43     RTI_INT32 address_bit_count;
44
45     /*e \dref_Transport_Property_t_properties_bitmap */
46     RTI_INT32 properties_bitmap;
47
48     /*e \dref_Transport_Property_t_gather_send_buffer_count_max */
49     RTI_INT32 gather_send_buffer_count_max;
50
51     /*e \dref_Transport_Property_t_message_size_max */
52     RTI_INT32 message_size_max;
53
54     /*e \dref_Transport_Property_t_allow_interfaces_list */
55     char **allow_interfaces_list;
56
57     /*e \dref_Transport_Property_t_allow_interfaces_list_length */
58     RTI_INT32 allow_interfaces_list_length;
```

```
59
60      /*e \dref_Transport_Property_t_deny_interfaces_list */
61      char **deny_interfaces_list;
62
63      /*e \dref_Transport_Property_t_deny_interfaces_list_length */
64      RTI_INT32 deny_interfaces_list_length;
65
66      /*e \dref_Transport_Property_t_allow_multicast_interfaces_list */
67      char **allow_multicast_interfaces_list;
68
69      /*e \dref_Transport_Property_t_allow_multicast_interfaces_list_length */
70      RTI_INT32 allow_multicast_interfaces_list_length;
71
72      /*e \dref_Transport_Property_t_deny_multicast_interfaces_list */
73      char **deny_multicast_interfaces_list;
74
75      /*e \dref_Transport_Property_t_deny_multicast_interfaces_list_length */
76      RTI_INT32 deny_multicast_interfaces_list_length;
77
78      /*e \dref_Transport_Property_t_transport_uuid */
79      struct NDDS_Transport_UUID transport_uuid;
80
81      /*i \dref_Transport_Property_t_domain_participant_ptr */
82      void *domain_participant_ptr;
83
84      /*i \dref_Transport_Property_t_domain_participant_forwarder */
85      const struct NDDS_Transport_Plugin_DomainParticipantForwarder
86              *domain_participant_forwarder;
87
88      /*e \dref_Transport_Property_t_thread_name_prefix */
89      char *thread_name_prefix;
90
91      /*e \dref_Transport_Property_t_max_interface_count */
92      RTI_INT32 max_interface_count;
93  };
94
95  /*i
96   * @brief Free the memory of the interface list property.
97   * @param allowInterfaceList \b In. The list of interfaces.
98   * @param allowInterfaceListLength \b In. The length of the list of interfaces.
99   */
100 extern NDDS_Transport_DllExport void
101 NDDS_Transport_Property_finalize_interface_list(
102         char ***allowInterfaceList,
103         RTI_INT32 *allowInterfaceListLength);
104
105 /*i
106  * \ingroup NDDS_TransportConfigurationComponent
107  *
108  * \brief  Safely copies NDDS_Transport_Property_t.
109  *
110  * @return 1 copy was successful, 0 if there is an error.
111 */
112 extern NDDS_Transport_DllExport RTIBool NDDS_Transport_Property_copy(
113         struct NDDS_Transport_Property_t *dst,
114         const struct NDDS_Transport_Property_t *src);
115
116 /*i
117  * \ingroup NDDS_TransportConfigurationComponent
118  * \brief Returns NDDS_Transport_Property_t arrays.
119  */
120 extern NDDS_Transport_DllExport void NDDS_Transport_Property_finalize(
121         struct NDDS_Transport_Property_t *property);
122
123 /*i
124  * \ingroup NDDS_TransportConfigurationComponent
125  *
126  * \brief Used to verify that a NDDS_Transport_Property_t is consistent
127  * and valid.
128  *
129  * @param property_in \st_in
130  *
131  * @return 1 if the stucture is self-consistent, 0 if there is an error.
132 */
133 extern NDDS_Transport_DllExport RTI_INT32 NDDS_Transport_Property_verify(
134         const struct NDDS_Transport_Property_t *property_in);
135
136 /*i
137  * \ingroup NDDS_TransportConfigurationComponent
138  * \brief This method returns an array containing the list of valid parent
139  * properties which are common for all the transports.
```

```
140  */
141 extern NDDS_Transport_DllExport const char **
142 NDDS_Transport_Property_get_common_properties(void);
143
144 /*i
145  * \ingroup NDDS_TransportConfigurationComponent
146  * @brief parse transport Class ID property
147  * @param property_value \b In. property value to parse.
148  * @param class_id \b Out. Output classId.
149  * @return RTI_TRUE if the parsing is done properly, RTI_FLASE otherwise.
150  */
151 extern NDDS_Transport_DllExport RTIBool NDDS_Transport_parse_transport_class_id(
152         const char *property_value,
153         NDDS_Transport_ClassId_t *class_id);
154
155 /*i
156  * \ingroup NDDS_TransportConfigurationComponent
157  *
158  * \brief Set polled or blocking behavior of transport.
159 *
160  * Some Transport Plugins may support the ability to have polled
161  * receive calls as well as blocking receive calls.  This bit allows
162  * the instantiator of the plugin to decide in which receive mode that
163  * the Transport Plugin instance will run.
164 *
165  * Currently, this ability is supported by the builtin Transport
166  * Plugins only for testing purposes.  Thus this bit is not exposed to
167  * customers for their own Transport Plugins.
168  */
169 #define NDDS_TRANSPORT_PROPERTY_BIT_POLLED (0x1)
170
171 /*e
172  * \ingroup NDDS_TransportConfigurationComponent
173  *
174  * \brief Specified zero-copy behavior of transport.
175  *
176  * A Transport Plugin may commit to one of three behaviors for zero copy
177  * receives:
178  *
179  * <OL>
180  * <LI>Always does zero copy.
181  * <LI>Sometimes does zero copy, up to the transport discretion.
182  * <LI>Never does zero copy.
183  * </OL>
184  *
185  * This bit should be set only if the Transport Plugin commits to
186  * always doing a zero copy receive, or more specifically, always
187  * loaning a buffer through its \c receive_rEA() call.
188  *
189  * In that case, \ndds will not need to allocate storage for a
190  * message that it retrieves with the \c receive_rEA() call.
191  *
192  * \internal
193  * @see \ref NDDS_TransportPlugin_loaning_receive
194  */
195 #define NDDS_TRANSPORT_PROPERTY_BIT_BUFFER_ALWAYS_LOANED (0x2)
196
197 /*e
198  * \ingroup NDDS_TransportConfigurationComponent
199  *
200  * \brief Minimum number of gather-send buffers that must be supported by a
201  * Transport Plugin implementation.
202  *
203  * For the NDDS_Transport_Property_t structure to be valid, the value
204  * of NDDS_Transport_Property_t#gather_send_buffer_count_max must be
205  * greater than or equal to this value.
206  *
207  * \internal
208  * @see \ref NDDS_TransportPlugin_gather_send
209  */
210 #define NDDS_TRANSPORT_PROPERTY_GATHER_SEND_BUFFER_COUNT_MIN (3)
211
212 #ifdef __cplusplus
213 } /* extern "C" */
214 #endif
215
216 #endif /* transport_interface_user_h */
```

10.5 transport_shmem.h

```
1  /*
2   * (c) Copyright, Real-Time Innovations, 2005-2024.
3   * All rights reserved.
4   *
5   * No duplications, whole or partial, manual or electronic, may be made
6   * without express written permission.  Any such copies, or
7   * revisions thereof, must display this notice unaltered.
8   * This code contains trade secrets of Real-Time Innovations, Inc.
9   */
10
11 #ifndef transport_shmem_h
12 #define transport_shmem_h
13
14 #include "clock/clock_interface.h"
15 #include "transport/transport_dll.h"
16 #include "transport/transport_interface.h"
17
18 #ifdef __cplusplus
19 extern "C" {
20 #endif
21
22 /* ================================================================= */
23 /*              Component: Shared Memory Transport Plugin            */
24 /* ================================================================= */
25
26 /*e \dref_ShmemTransportBuiltinGroupDocs
27  */
28
29 /*i \file
30  * \ingroup NDDS_Transport_Shmem_Plugin NDDS_TransportBuiltinsComponent
31  * \brief Shared-Memory Transport Plugin interface and definitions.
32  */
33
34 /* ================================================================= */
35 /*                 Shared Memory Properties                          */
36 /* ================================================================= */
37 /*e \dref_ShmemTransport_Property_t
38  */
39 struct NDDS_Transport_Shmem_Property_t {
40     /*e \dref_ShmemTransport_Property_t_parent
41      */
42     struct NDDS_Transport_Property_t parent;
43
44     /*e \dref_ShmemTransport_Property_t_received_message_count_max
45      */
46     RTI_INT32 received_message_count_max;
47
48     /*e \dref_ShmemTransport_Property_t_receive_buffer_size
49      */
50     RTI_INT32 receive_buffer_size;
51
52     /*e \dref_ShmemTransport_Property_t_enable_udp_debugging
53      */
54     RTIBool enable_udp_debugging;
55
56     /*e \dref_ShmemTransport_Property_t_udp_debugging_address
57      */
58     NDDS_Transport_Address_t udp_debugging_address;
59
60     /*e \dref_ShmemTransport_Property_t_udp_debugging_port
61      */
62     NDDS_Transport_Port_t udp_debugging_port;
63
64     /*i offset used for port -> segment key mapping */
65     RTI_INT32 segmentKeyOffset;
66
67     /*i factor used for port -> segment key mapping */
68     RTI_INT32 segmentKeyFactor;
69
70     /*i offset used for port -> semaphore key mapping */
71     RTI_INT32 semaphoreKeyOffset;
72
73     /*i factor used for port -> semaphore key mapping */
74     RTI_INT32 semaphoreKeyFactor;
75
76     /*i offset used for port -> mutex key mapping */
77     RTI_INT32 mutexKeyOffset;
78
```

```
79      /*i factor used for port -> mutex key mapping */
80      RTI_INT32 mutexKeyFactor;
81
82      /*i
83       * The maximum duration that a writer can take to write this message.
84       * If a writer takes longer, it can be potentially be aborted writing by a
85       * reader believing the writer is stuck (i.e. in an infinite loop).
86       * The resolution is in milliseconds.
87       * Default value is 1000
88       */
89      RTI_UINT32 maxAllowedWriterDurationMillisec;
90
91      /*i
92       * IMPORTANT: This value is used for unit testing purposes ONLY!. Possible
93       * values are:
94       * 1) If this value is
95       *    NDDS_TRANSPORT_SHMEM_RESERVED_FORCE_RECV_RESOURCE_FAILURE then create
96       *    recv resource will fail
97       * 2) If this value is
98       *    NDDS_TRANSPORT_SHMEM_RESERVED_USE_TRANSPORT_UUID_AS_SHMEM_UUID
99       *    then the shmemUuid will be the same as the transport_uuid.
100      */
101     RTIBool reserved;
102
103     /*i
104      * UUID used to check the reachability of a shmem locator received through
105      * discovery. After attaching to a shmem segment, we will check that
106      * the UUID in the header matches the one in the received locator.
107      * If it matches, that means that we are on the same machine than
108      * the target locator and that communication is possible through shmem.
109      */
110     NDDS_Transport_Address_t shmemUUID;
111
112     /*i
113      * Major version of the shared memory protocol.
114      * Different major versions are incompatible.
115      */
116     RTI_INT16 majorVersion;
117 };
118
119 /*i
120  * For testing: When reserved is set to this value, create_recv_resource will
121  * fail.
122  */
123 #define NDDS_TRANSPORT_SHMEM_RESERVED_FORCE_RECV_RESOURCE_FAILURE 1
124
125 /*i
126  * For testing: When reserved is set to this value,
127  * the shmemUuid will be the same as the transport_uuid.
128  *
129  * This can be used to simulate scenarios where two participants are on
130  * different machines even though they run on the same physical machine.
131  */
132 #define NDDS_TRANSPORT_SHMEM_RESERVED_USE_TRANSPORT_UUID_AS_SHMEM_UUID 2
133
134 /*i \ingroup NDDS_Transport_Shmem_Plugin
135  * \brief Shmem plugin class string name
136  */
137 #define NDDS_TRANSPORT_SHMEM_CLASS_NAME "shmem"
138
139 /*i \ingroup NDDS_Transport_Shmem_Plugin
140  * \brief Shmem plugin builtin alias string name
141  */
142 #define NDDS_TRANSPORT_BUILTIN_SHMEM_ALIAS "builtin.shmem"
143
144 /*e \ingroup  NDDS_Transport_Shmem_Plugin
145  * \brief Default value of NDDS_Transport_Property_t#address_bit_count
146  */
147 #define NDDS_TRANSPORT_SHMEM_ADDRESS_BIT_COUNT (-96)
148
149 /*e @ingroup NDDS_Transport_Shmem_Plugin
150  * @brief Default value of NDDS_Transport_Property_t#properties_bitmap.
151  */
152 #define NDDS_TRANSPORT_SHMEM_PROPERTIES_BITMAP_DEFAULT \
153   (NDDS_TRANSPORT_PROPERTY_BIT_BUFFER_ALWAYS_LOANED)
154
155 /*e \ingroup NDDS_Transport_Shmem_Plugin
156  * \brief Default value of
157  * NDDS_Transport_Property_t#gather_send_buffer_count_max.
158  */
159 #define NDDS_TRANSPORT_SHMEM_GATHER_SEND_BUFFER_COUNT_MAX_DEFAULT (1024)
```

```
160
161 /*e \ingroup NDDS_Transport_Shmem_Plugin
162  * \brief Default value of NDDS_Transport_Property_t#message_size_max.
163  */
164 #if defined(RTI_INTY)
165   #define NDDS_TRANSPORT_SHMEM_MESSAGE_SIZE_MAX_DEFAULT (9216)
166 #else
167   #define NDDS_TRANSPORT_SHMEM_MESSAGE_SIZE_MAX_DEFAULT (65536)
168 #endif
169
170 /*e \ingroup NDDS_Transport_Shmem_Plugin
171  * \brief Default value of
172  * NDDS_Transport_Shmem_Property_t#received_message_count_max.
173  */
174 #if defined(RTI_INTY)
175   #define NDDS_TRANSPORT_SHMEM_RECEIVED_MESSAGE_COUNT_MAX_DEFAULT (8)
176 #else
177   #define NDDS_TRANSPORT_SHMEM_RECEIVED_MESSAGE_COUNT_MAX_DEFAULT (64)
178 #endif
179
180 /*e \ingroup NDDS_Transport_Shmem_Plugin
181  * \brief Default value of NDDS_Transport_Shmem_Property_t#receive_buffer_size.
182  */
183 #define NDDS_TRANSPORT_SHMEM_RECEIVE_BUFFER_SIZE_DEFAULT \
184   (NDDS_TRANSPORT_SHMEM_RECEIVED_MESSAGE_COUNT_MAX_DEFAULT \
185    * NDDS_TRANSPORT_SHMEM_MESSAGE_SIZE_MAX_DEFAULT / 4)
186
187 /*i \ingroup NDDS_Transport_Shmem_Plugin
188  * \brief Major version of the shared memory transport containing bug 14240
189  * (RTI-28)
190  */
191 #define NDDS_TRANSPORT_SHMEM_MAJOR_BEFORE_BUG_14240_FIX (1)
192
193 /*e \ingroup NDDS_Transport_Shmem_Plugin
194  * \brief Major version for the transport plugin after fixing bug 14240 (RTI-28)
195  */
196 #define NDDS_TRANSPORT_SHMEM_MAJOR_AFTER_BUG_14240_FIX (2)
197
198 /* SparcSolaris had a shmem version with major 3 */
199 #define NDDS_TRANSPORT_SHMEM_MAJOR_SPARC_SOLARIS_DEPRECATED (3)
200
201 /*
202  * This is the version of the shared memory transport that uses a concurrent
203  * queue with a writer cookie, used in Network Capture (CORE-10280) to get the
204  * PID of the source for inbound traffic.
205  *
206  * There is a parallelism between concurrent queue versions and shared memory
207  * versions. We are doing the same here as in
208  * REDA_CONCURRENT_QUEUE_VERSION_AFTER_WRITER_COOKIE_SUPPORT: keep the default
209  * major version so that there is no interoperability break, but allow (through
210  * property configuration) to use a newer one.
211  */
212 #define NDDS_TRANSPORT_SHMEM_MAJOR_AFTER_WRITER_COOKIE_SUPPORT (4)
213
214 /*
215  * This is the version of the shared memory transport that uses a robust
216  * pthread mutex to protect the shared memory segment.
217  */
218 #define NDDS_TRANSPORT_SHMEM_MAJOR_AFTER_ROBUST_PTHREAD_MUTEX (5)
219
220 /* This is the current default major version. */
221 #define NDDS_TRANSPORT_SHMEM_VERSION_MAJOR_DEFAULT \
222   NDDS_TRANSPORT_SHMEM_MAJOR_AFTER_BUG_14240_FIX
223
224 /*e \ingroup NDDS_Transport_Shmem_Plugin
225  * \brief Use this to initialize stack variable.
226  * \hideinitializer
227  */
228 #define NDDS_TRANSPORT_SHMEM_PROPERTY_DEFAULT \
229     { \
230         { \
231             NDDS_TRANSPORT_CLASSID_SHMEM, \
232             NDDS_TRANSPORT_SHMEM_ADDRESS_BIT_COUNT, \
233             NDDS_TRANSPORT_SHMEM_PROPERTIES_BITMAP_DEFAULT, \
234             NDDS_TRANSPORT_SHMEM_GATHER_SEND_BUFFER_COUNT_MAX_DEFAULT, \
235             NDDS_TRANSPORT_SHMEM_MESSAGE_SIZE_MAX_DEFAULT, \
236             NULL, \
237             0, /* allow_interfaces_list */ \
238             NULL, \
239             0, /* deny_interfaces_list */ \
240             NULL, \
```

```
241             0, /* allow_multicast_interfaces_list */ \
242             NULL, \
243             0,                          /* deny_multicast_interfaces_list */ \
244             NDDS_TRANSPORT_UUID_UNKNOWN, /* transport_uuid */ \
245             NULL,                       /* domain_participant_ptr */ \
246             NULL,                       /* domain_participant_forwarder */ \
247             NULL,                       /* thread_name_prefix */ \
248             NDDS_TRANSPORT_LENGTH_UNLIMITED, /* max_interface_count */ \
249         }, \
250                 NDDS_TRANSPORT_SHMEM_RECEIVED_MESSAGE_COUNT_MAX_DEFAULT, \
251                 NDDS_TRANSPORT_SHMEM_RECEIVE_BUFFER_SIZE_DEFAULT, \
252                 RTI_FALSE, /* enable_udp_debugging */ \
253                 { \
254                     { 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 239, 255, 1, 2 } \
255                 },          /* udp_debugging_address */ \
256                 7399,       /* udp_debugging_port */ \
257                 0x400000,   /* 0100 xxxx ... */ \
258                 1, 0x800000, /* 1000 xxxx ... */ \
259                 1, 0xB00000, /* 1100 xxxx ... */ \
260                 1, 1000,    /* maxAllowedWriterDurationMillisec */ \
261                 RTI_FALSE,  /* reserved */ \
262                 NDDS_TRANSPORT_ADDRESS_INVALID_INITIALIZER, /* shmemUUID */ \
263                 NDDS_TRANSPORT_SHMEM_VERSION_MAJOR_DEFAULT  /* majorVersion */ \
264     }
265
266 /* ================================================================= */
267 /*                 Shared Memory Transport Class                     */
268 /* ================================================================= */
269
270 /*i \ingroup NDDS_Transport_Shmem_Plugin
271  * \brief Create a new shmem process transport.
272  *
273  * \copydoc NDDS_Transport_Shmem_new
274  *
275  * @exception RTI_LOG_MALLOC_FAILURE_d
276  * RTI_LOG_CREATION_FAILURE_s RTI_LOG_PRECONDITION_FAILURE
277  * RTI_LOG_INIT_FAILURE_s
278  */
279 extern NDDS_Transport_DllExport NDDS_Transport_Plugin *
280 NDDS_Transport_Shmem_newI(
281         const struct NDDS_Transport_Shmem_Property_t *property);
282
283 /*e \ingroup NDDS_Transport_Shmem_Plugin
284  * \brief Create a new shmem process transport.
285  *
286  * An application may create multiple transports, possibly for use in
287  * different domains.
288  *
289  * @param property_in \st_in Desired behavior of this transport. May be NULL
290  * for default property. The transport plugin can only support one unicast
291  * receive interface; therefore the interface selection lists are ignored.
292  *
293  * @return handle to a Shmem inter-process Transport Plugin on success
294  * @return NULL on failure.
295  */
296 extern NDDS_Transport_DllExport NDDS_Transport_Plugin *NDDS_Transport_Shmem_new(
297         const struct NDDS_Transport_Shmem_Property_t *property_in);
298
299 /*i \ingroup NDDS_Transport_Shmem_Plugin
300  */
301 #define NDDS_Transport_Shmem_new(property_in) \
302   NDDS_Transport_Shmem_newI(property_in)
303
304 /*i \ingroup NDDS_Transport_Shmem_Plugin
305  * \brief Realization of NDDS_Transport_Get_Class_Name_Fcn_cEA for Shmem
306  * transports.
307  *
308  * Returns the string "shmem".
309  *
310  * @see NDDS_Transport_Get_Class_Name_Fcn_cEA for complete
311  * documentation.
312  */
313 extern NDDS_Transport_DllExport const char *
314 NDDS_Transport_Shmem_get_class_name_cEA(NDDS_Transport_Plugin *self);
315
316 /*i \ingroup NDDS_Transport_Shmem_Plugin
317  * \brief Realization of NDDS_Transport_String_To_Address_Fcn_cEA for Shmem
318  * transports.
319  *
320  * There is no natural string representation for an "Shmem"
321  * transport address.  Will always return 0.
```

```
322  *
323  * @param address_out \st_out The corresponding numerical value in IPv6
324  * format.
325  *
326  * @param address_in \st_in
327  *
328  * @see NDDS_Transport_String_To_Address_Fcn_cEA for complete
329  * documentation.
330  */
331 extern NDDS_Transport_DllExport RTI_INT32
332 NDDS_Transport_Shmem_string_to_address_cEA(
333         NDDS_Transport_Plugin *self,
334         NDDS_Transport_Address_t *address_out,
335         const char *address_in);
336
337 /*i \ingroup NDDS_Transport_Shmem_Plugin
338  * \brief Realization of NDDS_Transport_Get_Receive_Interfaces_Fcn_cEA
339  * for Shmem transports.
340  *
341  * @see NDDS_Transport_Get_Receive_Interfaces_Fcn_cEA for complete
342  * documentation.
343  *
344  */
345 extern NDDS_Transport_DllExport RTI_INT32
346 NDDS_Transport_Shmem_get_receive_interfaces_cEA(
347         NDDS_Transport_Plugin *self,
348         RTI_INT32 *found_more_than_provided_for_out,
349         RTI_INT32 *interface_reported_count_out,
350         NDDS_Transport_Interface_t interface_array_inout[],
351         RTI_INT32 interface_array_size_in);
352
353 /*i \ingroup NDDS_Transport_Shmem_Plugin
354  * \brief Returns true if the Shared Memory plugin is usable, false if
355  * otherwise.
356  *
357  * The shared memory plugin may be unusable if a call to create_recv_resource
358  * fails to allocate the necessary shared memory segments due to system resource
359  * constraints.
360  */
361 extern NDDS_Transport_DllExport RTIBool
362 NDDS_Transport_Shmem_is_valid(NDDS_Transport_Plugin *self);
363
364 /*i \ingroup NDDS_Transport_Shmem_Plugin
365  * \brief Checks if a shared memory locator is reachable from the current shared
366  * memory plugin.
367  *
368  * @param self \st_out Indicates if the SHMEM segment associated with port_in
369  * exists.
370  * @param s_reachable_out \st_out RTI_TRUE if the locator is reachable
371  * RTI_FALSE otherwise.
372  *
373  * @param address_in \st_in Input address expected to reach
374  *
375  * @param port_in \st_in Port in the address
376  */
377 extern NDDS_Transport_DllExport void
378 NDDS_Transport_Shmem_check_locator_reachability(
379         const NDDS_Transport_Plugin *self,
380         RTIBool *shmem_segment_exists,
381         RTIBool *is_reachable_out,
382         const NDDS_Transport_Address_t *address_in,
383         NDDS_Transport_Port_t port_in);
384
385 #ifdef __cplusplus
386 } /* extern "C" */
387 #endif
388
389 #endif /* transport_shmem_h */
```

10.6 transport_udp.h

```
1 /*
2  * (c) Copyright, Real-Time Innovations, 2005-2024.
3  * All rights reserved.
4  *
5  * No duplications, whole or partial, manual or electronic, may be made
6  * without express written permission.  Any such copies, or
```

```
7  * revisions thereof, must display this notice unaltered.
8  * This code contains trade secrets of Real-Time Innovations, Inc.
9  */
10
11 #ifndef transport_udp_h
12 #define transport_udp_h
13
14 #include "osapi/osapi_socket.h"
15 #include "osapi/osapi_atomic.h"
16 #include "clock/clock_interface.h"
17 #include "transport/transport_dll.h"
18 #include "transport/transport_interface.h"
19 #include "transport/transport_udp_wan_library.h"
20
21 #ifdef __cplusplus
22 extern "C" {
23 #endif
24
25 struct NDDS_Transport_UDP; /* forward declare */
26
27 struct NDDS_Transport_UDP_SocketFactory;
28
29 struct NDDS_Transport_UDP_Interface_t;
30
31 /* ========================================================================= */
32 /*                 Component: UDP/IP Transport Plugin                        */
33 /* ========================================================================= */
34
35 /*e \dref_UDPTransportBuiltinGroupDocs
36  */
37
38 /*i \file
39  *   \ingroup NDDS_Transport_UDP_Plugin
40  *   \brief UDP Transport Plugin interfaces and definitions.
41  */
42
43 /* ========================================================================= */
44 /*             Transport UDP WAN Listener Class                              */
45 /* ========================================================================= */
56 typedef void (
57         *NDDS_Transport_UDP_WAN_Listener_Public_Mapping_Resolved_Callback)(
58         void *listener_data,
59         const NDDS_Transport_Address_t *public_uuid_adress,
60         NDDS_Transport_Port_t rtps_port);
61
62 struct NDDS_Transport_UDP_WAN_Listener {
63     void *listener_data;
64     NDDS_Transport_UDP_WAN_Listener_Public_Mapping_Resolved_Callback
65             on_public_mapping_resolved;
66 };
67
68 /* ========================================================================= */
69 /*                  UDP Port                                                 */
70 /* ========================================================================= */
71 /*e \dref_UDPTransport_Port
72  */
73 typedef RTI_UINT16 NDDS_Transport_UDP_Port;
74
75 #define NDDS_TRANSPORT_UDP_PORT_INVALID (0)
76 #define NDDS_TRANSPORT_UDP_PORT_MAX (RTI_UINT16_MAX)
77
78 /* ========================================================================= */
79 /*                  UDP WAN CommPorts                                        */
80 /* ========================================================================= */
81
82 /*e \dref_UDPTransport_WAN_CommPortsMappingInfo
83  */
84 struct NDDS_Transport_UDP_WAN_CommPortsMappingInfo {
85     /*e \dref_UDPTransport_WAN_CommPortsMappingInfo_rtps_port
86      */
87     NDDS_Transport_Port_t rtps_port;
88     /*e \dref_UDPTransport_WAN_CommPortsMappingInfo_host_port
89      */
90     NDDS_Transport_UDP_Port host_port;
91     /*e \dref_UDPTransport_WAN_CommPortsMappingInfo_public_port
92      */
93     NDDS_Transport_UDP_Port public_port;
94     /*i
95      * CORE-16439: It is possible the before a receive resource has been removed
96      * (e.g, interface goes DOWN) the application tries to recreate it again
97      * (e.g, interfaces goes UP). In this scenario, if the
```

```
98      * previous resource is still in the process of being removed the creation
99      * of the new resource will fail because it will try to add the same mapping
100      * again to the comm_ports_list. To avoid this situation, we keep a
101      * reference count of how many receive resources are using this mapping
102      * info.
103      *
104      * See CORE-16439 for additional details.
105      *
106      * This field is marked as internal as it is not intended to be used
107      * by applications.
108      */
109     int ref_count;
110 };
111 
112 /* ================================================================= */
113 /*                  Unicast UDP Properties                           */
114 /* ================================================================= */
115 
116 struct NDDS_Transport_UDP_Property_t {
117     struct NDDS_Transport_Property_t parent;
118     RTI_INT32 send_socket_buffer_size;
119     RTI_INT32 recv_socket_buffer_size;
120     RTI_INT32 unicast_enabled;
121     RTI_INT32 multicast_enabled;
122     RTI_INT32 multicast_ttl;
123     RTI_INT32 multicast_loopback_disabled;
124     RTI_INT32 ignore_loopback_interface;
125     RTI_INT32 ignore_nonup_interfaces;
126     RTI_INT32 ignore_nonrunning_interfaces;
127     RTI_INT32 no_zero_copy;
128     RTI_INT32 send_blocking;
129     RTI_INT32 enable_v4mapped;
130     RTI_INT32 use_checksum;
131     RTI_UINT32 transport_priority_mask;
132     RTI_INT32 transport_priority_mapping_low;
133     RTI_INT32 transport_priority_mapping_high;
134     RTI_INT32 send_ping;
135     RTI_INT32 force_interface_poll_detection;
136     RTI_UINT32 interface_poll_period;
137     RTI_INT32 reuse_multicast_receive_resource;
138     RTI_INT32 protocol_overhead_max;
139     RTI_INT32 disable_interface_tracking;
140     RTI_UINT32 join_multicast_group_timeout;
141     char *public_address;
142     struct NDDS_Transport_UDP_WAN_CommPortsMappingInfo *comm_ports_list;
143     RTI_INT32 comm_ports_list_length;
144     struct NDDS_Transport_UDP_WAN_Listener *wan_listener;
145     RTI_INT32 port_offset;
146     RTI_UINT32 binding_ping_period;
147     /*
148      * Maximum time that an orphan bindingping mapping may exist without being
149      * refreshed before it is removed.
150      */
151     RTI_UINT32 binding_ping_mapping_expiration;
152     /*i
153      * \brief If this property is set, we use this function to check for an
154      * enabled WAN transport. If the function is not set, we would try to load a
155      * default shared library with the correct function to enable WAN transport.
156      */
157     NDDS_Transport_UDP_WAN_Library_Is_Plugin_Enabled_Function
158             plugin_enabled_function_ptr;
159     /****************** START TESTING ONLY FIELDS *****************/
160     /*i
161      * \brief If this property is set we will update the send
162      * and receive statistics split into participant and endpoint discovery
163      * by parsing the packets. We will update the discovery/user packets
164      * based on the port number regardless of if this property is set.
165      * The default is false for performance reasons and is set to true only
166      * for testing purposes.
167      */
168     RTIBool gather_detailed_statistics;
169     /*i
170      * \brief If this property is set to TRUE we will check if the guid prefix
171      * in the RTPS header matches guid_prefix_to_drop. If it does we will drop
172      * the sample.
173      */
174     RTIBool drop_samples_by_guid_prefix;
175     /*i
176      * \brief GUID prefix to drop when drop_samples_by_guid_prefix is set to
177      * TRUE.
178      */
```

```
179     struct RTIOsapiRtpsGuidPrefix guid_prefix_to_drop;
180     /****************** END TESTING ONLY FIELDS *****************/
181 };
182
183 /*e \ingroup NDDS_Transport_UDP_Plugin
184  * \brief Default value of NDDS_Transport_Property_t#properties_bitmap.
185  */
186 #define NDDS_TRANSPORT_UDP_PROPERTIES_BITMAP_DEFAULT (0)
187
188 /*e \ingroup NDDS_Transport_UDP_Plugin
189  * \brief Default value of
190  * NDDS_Transport_Property_t#gather_send_buffer_count_max.
191  *
192  * This is also the maximum value that can be used when instantiating the udp
193  * transport.
194  *
195  * 16 is sufficient for \ndds, but more may improve discovery and reliable
196  * performance.
197  * Porting note: find out what the maximum gather buffer count is on your OS!
198  */
199 #define NDDS_TRANSPORT_UDP_GATHER_SEND_BUFFER_COUNT_MAX_DEFAULT (16)
200
201 /*e \ingroup NDDS_Transport_UDP_Plugin
202  * \brief Used to specify that os maximum be used to specify socket buffer size.
203  *
204  * This property, if set, uses the maximum allowable size set in your OS for
205  * send and receive socket buffer sizes. If you update the maximum allowable
206  * size in your OS setting, then \ndds will automatically pick up that change.
207  * This property is not supported on all platforms because not all platforms
208  * have maximum values for the size of the send and receive buffers for sockets.
209  * If the platform does not support this property, then the property will be
210  * set to \ref
211  * NDDS_Transport_UDP_Plugin#NDDS_TRANSPORT_UDP_SEND_SOCKET_BUFFER_SIZE_DEFAULT
212  * or \ref
213  * NDDS_Transport_UDP_Plugin#NDDS_TRANSPORT_UDP_RECV_SOCKET_BUFFER_SIZE_DEFAULT,
214  * depending on the property being set.
215  * See the Platform Notes to determine if this property is supported on your
216  * platform.
217  *
218  * This value is -2.
219  */
220 #define NDDS_TRANSPORT_UDP_SOCKET_BUFFER_SIZE_OS_MAX \
221     RTI_OSAPI_UDP_SOCKET_BUFFER_SIZE_OS_MAX
222
223 /*e \ingroup NDDS_Transport_UDP_Plugin
224  * \brief Used to specify that os default be used to specify socket buffer size.
225  *
226  * This property, if set, uses the default size set in your OS for send and
227  * receive socket buffer sizes. If you update the default size in your OS
228  * settings, then \ndds will automatically pick up that change.
229  *
230  * This value is -1.
231  */
232 #define NDDS_TRANSPORT_UDP_SOCKET_BUFFER_SIZE_OS_DEFAULT \
233     RTI_OSAPI_UDP_SOCKET_BUFFER_SIZE_OS_DEFAULT
234
235 /*e \ingroup NDDS_Transport_UDP_Plugin
236  * \brief Default value of send_socket_buffer_size.
237  *
238  * The default value is 2097152 bytes (1048576 bytes on AIX platforms).
239  */
240 #define NDDS_TRANSPORT_UDP_SEND_SOCKET_BUFFER_SIZE_DEFAULT \
241     RTI_OSAPI_UDP_SEND_SOCKET_BUFFER_SIZE_DEFAULT
242
243 /*e \ingroup NDDS_Transport_UDP_Plugin
244  * \brief Default value of recv_socket_buffer_size.
245  *
246  * The default value is 2097152 bytes (1048576 bytes on AIX platforms).
247  */
248 #define NDDS_TRANSPORT_UDP_RECV_SOCKET_BUFFER_SIZE_DEFAULT \
249     RTI_OSAPI_UDP_RECV_SOCKET_BUFFER_SIZE_DEFAULT
250
251 #define NDDS_TRANSPORT_UDP_DATAGRAM_LENGTH_MAX (65535)
252
253 /*e \ingroup NDDS_Transport_UDP_Plugin
254  * \brief Default value of multicast_ttl.
255  */
256 #define NDDS_TRANSPORT_UDP_MULTICAST_TTL_DEFAULT (1)
257
258
259 #ifdef RTI_MULTICAST
```

```
260   #define NDDS_TRANSPORT_UDP_USE_MULTICAST_DEFAULT 1
261 #else
262   #define NDDS_TRANSPORT_UDP_USE_MULTICAST_DEFAULT 0
263 #endif
264
265
266 /*e \dref_UDPTransport_IGNORE_LOOPBACK_DISABLED
267  */
268 #define NDDS_TRANSPORT_UDP_IGNORE_LOOPBACK_DISABLED (0)
269
270 /*e \dref_UDPTransport_IGNORE_LOOPBACK_ENABLED
271  */
272 #define NDDS_TRANSPORT_UDP_IGNORE_LOOPBACK_ENABLED (1)
273
274 /*e \dref_UDPTransport_IGNORE_LOOPBACK_AUTO
275  */
276 #define NDDS_TRANSPORT_UDP_IGNORE_LOOPBACK_AUTO (-1)
277
278
279 /*e \dref_UDPTransport_BLOCKING_NEVER
280  */
281 #define NDDS_TRANSPORT_UDP_BLOCKING_NEVER (0)
282
283 /*e \dref_UDPTransport_BLOCKING_ALWAYS
284  */
285 #define NDDS_TRANSPORT_UDP_BLOCKING_ALWAYS (1)
286 #define NDDS_TRANSPORT_UDP_BLOCKING_UNICAST_ONLY (2)
287
288 /*e \dref_UDPTransport_BLOCKING_DEFAULT
289  */
290 #define NDDS_TRANSPORT_UDP_BLOCKING_DEFAULT NDDS_TRANSPORT_UDP_BLOCKING_ALWAYS
291
292 /* ================================================================= */
293 /*                   InterfaceListener Class                         */
294 /* ================================================================= */
295
296 /*i
297  * \defgroup NDDS_Transport_UDP_InterfaceListenerClass
298  * NDDS_Transport_UDP_InterfaceListener \ingroup NDDS_Transport_UDP_Plugin
299  * \brief Interface for IPv4 or v6 interface acceptance/rejection
300  *
301  * @mtsafety See NDDS_Transport_UDP_public UML diagram
302  *
303  * @see NDDS_Transport_UDP_newI
304  */
305 struct NDDS_Transport_UDP_InterfaceListener; /*forward declare */
306
307 /*i \ingroup NDDS_Transport_UDP_InterfaceListenerClass
308  * \brief A user-provided function to be used to filter out particular
309  * network interfaces to be used by the Transport Plugin.
310  *
311  * The user may implement this function and pass it to an
312  * instance of a Transport Plugin to reject or accept network
313  * interfaces that the plugin offers as ones to use to receive
314  * messages.
315  *
316  * When \ndds contacts other applications, it will provide a list of
317  * interfaces that other applications can use to send messages (on a
318  * plugin by plugin basis).  The addresses of the interfaces are used
319  * as the destination addresses of messages that are sent.  If a plugin
320  * rejects one of the available network interfaces, then that interface
321  * will not be offered as one to which other plugins should send
322  * messages.
323  *
324  * For interfaces that can be used by the plugin, this function also
325  * enables the user to select which interfaces \ndds is allowed to use
326  * in sending multicast messages.  The set of interfaces enabled for
327  * multicast sending is a subset of the interfaces that can be used for
328  * receiving.  That is, an interface cannot be enabled to send
329  * multicast messages but disabled for receiving unicast messages.
330  *
331  * Only IP network interfaces that are "UP" (enabled by the OS and
332  * ready to use) will be passed to this filter function.  The ones that
333  * are "DOWN" will not be used by definition.
334  *
335  * @param plugin \st_in The instance of the plugin for which the filter
336  * is being called.
337  *
338  * @param multicast_enabled_out \st_out The filter should set whether
339  * or not the plugin is allowed to use this interface for
340  * sending/receiving multicast messages.
```

```
341  *
342  * @param interface_in \st_in An interface offered by the plugin as a
343  * candidate for receive messages.
344  *
345  * @return 1 to use the interface.
346  *         \c multicast_enabled_out will indicate whether it is also OK
347  *         to use the interface to send multicast messages.
348  *
349  * @return 0 to reject using the interface.
350  */
351 typedef RTI_INT32 (*NDDS_Transport_UDP_Interface_Filter_Fcn)(
352         struct NDDS_Transport_UDP_InterfaceListener *self,
353         struct NDDS_Transport_UDP *plugin,
354         RTI_INT32 *multicast_enabled_out,
355         NDDS_Transport_Interface_t *interface_in,
356         const char *interface_name_in);
357
358 /*i \ingroup NDDS_Transport_UDP_InterfaceListenerClass
359  * \brief Pure virtual class for the interface listener.
360  * RTI does NOT supply a default interface listener
361  */
362 struct NDDS_Transport_UDP_InterfaceListener {
363     /* \copydoc NDDS_Transport_UDP_Interface_Filter_Fcn */
364     NDDS_Transport_UDP_Interface_Filter_Fcn onInterface;
365 };
366
367 /* ================================================================= */
368 /*                   Socket Factory Class                            */
369 /* ================================================================= */
370 /*i
371  * \defgroup NDDS_Transport_UDP_SocketFactoryClass
372  * NDDS_Transport_UDP_SocketFactory \ingroup NDDS_Transport_UDP_Plugin \brief
373  * Interface for socket creation and destruction.
374  *
375  * @mtsafety See NDDS_Transport_UDP_public UML diagram
376  *
377  * @see NDDS_Transport_UDP_new
378  */
379
380 /*i
381  * \ingroup NDDS_Transport_UDP_SocketFactoryClass
382  * \brief Interface representation for socket creation and destruction.
383  */
384 struct NDDS_Transport_UDP_SocketFactory;
385
386 /*i
387  * \ingroup NDDS_Transport_UDP_SocketFactoryClass
388  * \brief Create a new sending socket for the Itransport.
389  *
390  * @pre a valid UDP transport.
391  *
392  * @post A valid UDP send socket, that may not necessarily be bound.
393  *
394  * @param transport_in \b In.  The NDDS_Transport_UDP transport that will use
395  * the new socket.  Specific property of the transport, such as the maximum
396  * message size allowed, can be queried from the
397  * NDDS_Transport_Plugin#property
398  *
399  * @param interface_address_in \b In. If set, must be the address of
400  * a valid interface.  In addition, the socket created should be enabled
401  * to send via multicast.  If is NULL, then the socket created will not
402  * be enabled to send multicast.
403  *
404  * @param interface_index_in \b In. If set, must be the index of
405  * a valid interface.  In addition, the socket created should be enabled
406  * to send via multicast.  If is NULL, then the socket created will not
407  * be enabled to send multicast. Interface index is used instead of an
408  * interface address in UDPv6.
409  *
410  * @return Newly created socket descriptor.
411  *
412  * @mtsafety «oneThreadAtATime»
413  *
414  * @see NDDS_Transport_UDP_SocketFactory_create_send_socket
415  * NDDS_Transport_UDP_SocketFactory_Socket_Destroy_Fcn
416  */
417 typedef RTI_INT32 (*NDDS_Transport_UDP_SocketFactory_Send_Socket_Create_Fcn)(
418         struct NDDS_Transport_UDP_SocketFactory *me,
419         NDDS_Transport_Port_t *port_out,
420         const struct NDDS_Transport_UDP *transport_in,
421         const NDDS_Transport_Address_t *interface_address_in,
```

```
422         const int *interface_index_in);
423
424 /*i
425  * \ingroup NDDS_Transport_UDP_SocketFactoryClass
426  * \brief Create a new receiving socket for the transport, which
427  * may be a multicast transport.
428  *
429  * @pre valid IP unicast transport.
430  *
431  * @post A valid, bound unicast receive socket.
432  *
433  * @param port_inout. \b InOut. NOT optional. If
434  * NDDS_TRANSPORT_PORT_INVALID, a new, previously unassigned port must
435  * be chosen, and the port must be assigned a new value so the caller
436  * knows which port it is listening on.  If the port is a valid value,
437  * you must attempt to bind to that port, and not proceed in case of
438  * failure (such as if the port is already bound).
439  *
440  * @param transport_in \b In. The transport that wishes to use the
441  * socket.  Properties of the IP unicast transport described in
442  * NDDS_Transport_UDP_Property_t must be met in "at least" manner.  That
443  * is, doing more than requested is allowed (but will lead to waste
444  * unless the implementer has other insights which suggests more
445  * optimal use of resources), but creation of a socket that cannot meet
446  * the minimum requirement will put the system into an inconsistent
447  * state.  For example, the receiving socket will be either polled or
448  * blocking; if the transport is polled
449  * (NDDS_Transport_Plugin_isPolled) it is an error to create a blocking
450  * socket.  On the other hand, creating a socket that can buffer more
451  * data than requires is allowed.  Specific property of the transport,
452  * such as the maximum message size to be buffered, can be queried from
453  * the NDDS_Transport_Plugin#property.
454  *
455  * @param transport_priority_in \b In. transport_priority hint.  The
456  * RTI supplied socket factory NDDS_Transport_UDP_SocketFactory does not
457  * act on transport_priority.  If you want to implement a
458  * transport_priority aware socket factory, either implement your own
459  * from scratch or use
460  * NDDS_Transport_UDP_SocketFactory_createReceiveSocket and specialize.
461  *
462  * @param multicast_address_in \b In. Valid if joining a multicast group.
463  *
464  * @see NDDS_Transport_UDP_SocketFactory_Send_Socket_Create_Fcn
465  * NDDS_Transport_UDP_SocketFactory_create_receive_socket
466  * NDDS_Transport_UDP_SocketFactory_Socket_Destroy_Fcn
467  */
468 typedef RTI_INT32 (*NDDS_Transport_UDP_SocketFactory_Receive_Socket_Create_Fcn)(
469         struct NDDS_Transport_UDP_SocketFactory *me,
470         NDDS_Transport_Port_t *port_inout,
471         const struct NDDS_Transport_UDP *transport_in,
472         const NDDS_Transport_Address_t *multicast_address_in);
473
474 /*i \ingroup NDDS_Transport_UDP_SocketFactoryClass
475  * \brief Destroy a previously created socket.
476  *
477  * @pre Socket no longer used, and no thread blocked on the socket.
478  * @post All resources associate with the socket released.
479  *
480  * @param s_in \b In. socket descriptor.
481  *
482  * @mtsafety «oneThreadAtATime»
483  *
484  * @see NDDS_Transport_UDP_SocketFactory_destroy_socket
485  */
486 typedef RTI_INT32 (*NDDS_Transport_UDP_SocketFactory_Socket_Destroy_Fcn)(
487         struct NDDS_Transport_UDP_SocketFactory *me,
488         RTI_INT32 s_in);
489
490 /*i \ingroup NDDS_Transport_UDP_SocketFactoryClass
491  * \brief Pure virtual class for the socket factory functionality.
492  * RTI supplies the UDP socket factory.
493  */
494 struct NDDS_Transport_UDP_SocketFactory {
495     /*e Children: implement this method. */
496     NDDS_Transport_UDP_SocketFactory_Send_Socket_Create_Fcn create_send_socket;
497
498     /*e Children: implement this method. */
499     NDDS_Transport_UDP_SocketFactory_Receive_Socket_Create_Fcn
500             create_receive_socket;
501
502     /*e Children: implement this method to destroy both the send and
```

```
503      receive sockets. */
504     NDDS_Transport_UDP_SocketFactory_Socket_Destroy_Fcn destroy_socket;
505 };
506
507 /*i \ingroup NDDS_Transport_UDP_SocketFactoryClass
508  * \brief Convenience to check if the pure virtual methods have all been
509  * implemented
510  */
511 #define NDDS_Transport_UDP_SocketFactory_is_valid(m) \
512   ((m)->create_send_socket && (m)->create_receive_socket && (m)->destroy_socket)
513
514 /* ================================================================= */
515 /*                 IP Socket Factory Class                           */
516 /* ================================================================= */
517
518 /*i \ingroup NDDS_Transport_UDP_Plugin
519  * \brief Realization of NDDS_Transport_Get_Class_Name_Fcn_cEA for IPv6
520  * transports.
521  *
522  * Returns the string "udpv4" or "udpv6".
523  *
524  * @see NDDS_Transport_Get_Class_Name_Fcn_cEA for complete
525  * documentation.
526  */
527 extern NDDS_Transport_DllExport const char *
528 NDDS_Transport_UDP_get_class_name_cEA(NDDS_Transport_Plugin *self);
529
530 /*i \ingroup NDDS_Transport_UDP_Plugin
531  * \brief Realization of NDDS_Transport_String_To_Address_Fcn_cEA for IP
532  * transports.
533  *
534  * Converts a host name string to an address.
535  *
536  * @param self  NOT USED. May be NULL.
537  *
538  * @param address_out \st_out The corresponding numerical value in IPv6
539  * format.
540  *
541  * @param address_in \st_in The name of the address. It can be a
542  * dot notation name or a host name. If NULL, then the IP address of
543  * the localhost will be returned.
544  *
545  * @see NDDS_Transport_String_To_Address_Fcn_cEA for complete
546  * documentation.
547  */
548 extern NDDS_Transport_DllExport RTI_INT32
549 NDDS_Transport_UDP_string_to_address_cEA(
550         NDDS_Transport_Plugin *self,
551         NDDS_Transport_Address_t *address_out,
552         const char *address_in);
553
554 /* ================================================================= */
555 /*                     UDP/IP Transport Plugin                       */
556 /* ================================================================= */
557
558 /*i \ingroup NDDS_Transport_UDP_Plugin
559  * \brief Create an instance of a UDP Transport Plugin.
560  *
561  * An application may create and register multiple instances of this
562  * Transport Plugin with \ndds. Multiple instances of the plugin may be useful
563  * for partitioning the network interfaces across multiple \ndds domains.
564  * However, note that the underlying transport, the operating system's IP layer,
565  * is still a "singleton."  For example, if a unicast transport has already
566  * bound to a port, and another unicast transport tries to bind to the same
567  * port, the second attempt will fail.
568  *
569  * The transport plugin honors the interface/multicast "white" and "black"
570  * lists specified in the NDDS_Transport_UDP_Property_t#parent:
571  * - NDDS_Transport_Property_t#allow_interfaces_list,
572  * - NDDS_Transport_Property_t#deny_interfaces_list,
573  * - NDDS_Transport_Property_t#allow_multicast_interfaces_list,
574  * - NDDS_Transport_Property_t#deny_multicast_interfaces_list
575  *
576  * The format of a string in these lists is assumed to be in standard IPv4
577  * dot notation, possibly containing wildcards. For example:
578  *       - 10.10.30.232
579  *       - 10.10.*.*
580  *       - 192.168.1.*
581  *       - etc.
582  * Strings not in the correct format will be ignored.
583  *
```

```
584  * @param property_in \st_in Desired behavior of this transport. May be
585  * NULL for default property.
586  *
587  * @return A UDP Transport Plugin instance on success; or NULL on failure.
588  */
589 extern NDDS_Transport_DllExport NDDS_Transport_Plugin *NDDS_Transport_UDP_new(
590         const struct NDDS_Transport_UDP_Property_t *property_in);
591
592 /*i \ingroup NDDS_Transport_UDP_Plugin
593  * \brief Create an instance of a UDP Transport Plugin.
594  *
595  * \copydoc NDDS_Transport_UDP_new
596  *
597  * @param socket_factory_in \st_in The factory that will create all the
598  * sockets to be used in the transport.  May be NULL if default factory
599  * is to be used.  Only used by RTI internal testing.
600  * End users should set this parameter to NULL.
601  *
602  * @param interface_filter_in \st_in A function that will be used to
603  * filter out network interfaces that this plugin can use. May be NULL if
604  * default interface listener is to be used. The default listener uses
605  * The NDDS_Transport_UDP_Property_t#parent's lists to select the interfaces.
606  * Only used by RTI internal testing. End users should set this
607  * parameter to NULL.
608  *
609  * @exception RTI_LOG_MALLOC_FAILURE_d RTI_LOG_CREATION_FAILURE_s
610  * RTI_LOG_PRECONDITION_FAILURE RTI_LOG_INIT_FAILURE_s
611  */
612 extern NDDS_Transport_DllExport NDDS_Transport_Plugin *NDDS_Transport_UDP_newI(
613         const struct NDDS_Transport_UDP_Property_t *property_in,
614         struct NDDS_Transport_UDP_SocketFactory *socket_factory_in,
615         struct NDDS_Transport_UDP_InterfaceListener *interface_listener_in);
616
617 /*i \ingroup NDDS_Transport_UDP_Plugin
618  * \brief Realization of NDDS_Transport_Get_Receive_Interfaces_Fcn_cEA
619  * for IP transports.
620  *
621  * Retrieves a list of available network interfaces.
622  *
623  * @see NDDS_Transport_Get_Receive_Interfaces_Fcn_cEA for complete
624  * documentation.
625  */
626 extern NDDS_Transport_DllExport RTI_INT32
627 NDDS_Transport_UDP_get_receive_interfaces_cEA(
628         NDDS_Transport_Plugin *self,
629         RTI_INT32 *found_more_than_provided_for_out,
630         RTI_INT32 *interface_reported_count_out,
631         NDDS_Transport_Interface_t interface_array_inout[],
632         RTI_INT32 interface_array_size_in);
633
634 /*i \ingroup NDDS_Transport_UDP_Plugin
635  * \brief Retrieve ignore_loopback property
636  */
637 extern NDDS_Transport_DllExport RTI_INT32
638 NDDS_Transport_UDP_get_ignore_loopback_interface_status(
639         const NDDS_Transport_Plugin *self);
640
641 /*i \ingroup NDDS_Transport_UDP_Plugin
642  * \brief Checks if the provided address is a loopback address or a known
643  * interface address.
644  *
645  * The provided address is an IPv6 address which is internally mapped into a
646  * IPv4 socket address (network byte order).
647  *
648  * Unions are used to force 4-byte allignment of the IPV4 portion of
649  * the addresses. Otherwise we risk bus faults if we are ever passed
650  * unaligned address arrays.
651  */
652 extern NDDS_Transport_DllExport RTIBool NDDS_Transport_UDP_is_local_address(
653         const NDDS_Transport_Plugin *self,
654         const NDDS_Transport_Address_t *address_in);
655
656 /*i \ingroup NDDS_Transport_UDP_Plugin
657  *  Map transport_priority to an IP_TOS value based on the configured
658  *  transport properties.
659  *
660  *  @brief transport_priority to IP_TOS mapping
661  */
662 extern NDDS_Transport_DllExport RTI_INT32
663 NDDS_Transport_UDP_map_transport_priority(
664         struct NDDS_Transport_UDP_Property_t *property,
```

```
665         RTI_INT32 transport_priority_in);
666
667 /*i \ingroup NDDS_Transport_UDP_Plugin
668  * \brief Size of special "unblock" message
669  *
670  * When an ReceiveResource is unblocked with
671  * NDDS_Transport_unblock_recv_fnc_rrEA, a special unblock
672  * message may be sent. This message is guaranteed to be smaller than the
673  * minimum RTPS message size.  Higher layer that processes messages
674  * returned by netio should throw away messages of this size, as it was
675  * intended to unblock a blocked thread only.
676  */
677 #define NDDS_TRANSPORT_UDP_UNBLOCK_MESSAGE_SIZE \
678   (4 + 4 + sizeof(NDDS_Transport_Plugin *))
679
680 /*i \ingroup NDDS_Transport_UDP_Plugin
681  * \brief DiffServ max value
682  *
683  * Maximum value of the DiffServ field in an IP packet
684  */
685 #define NDDS_TRANSPORT_UDP_DIFF_SERV_MAX_VALUE RTI_OSAPI_SOCKET_MAX_DIFF_SERV
686
687 struct NDDS_Transport_UDP_Statistics {
688     /***********************
689      *   PUBLIC FIELDS
690      ***********************/
691     RTI_ATOMIC(RTI_UINT64) user_bytes_sent;
692     RTI_ATOMIC(RTI_UINT64) user_bytes_received;
693     RTI_ATOMIC(RTI_UINT64) user_packets_sent;
694     RTI_ATOMIC(RTI_UINT64) user_packets_received;
695     /*
696      * discovery_bytes and discovery_packets are calculated using the
697      * port number that is sent to/received from. This assumes the port
698      * is an even number. When using an odd port number (ie, using an odd
699      * domain ID), these statistics will not be calculated correctly.
700      * You should not use these members in tests where the domain ID
701      * and port numbers may change.
702      */
703     RTI_ATOMIC(RTI_UINT64) discovery_bytes_sent;
704     RTI_ATOMIC(RTI_UINT64) discovery_bytes_received;
705     RTI_ATOMIC(RTI_UINT64) discovery_packets_sent;
706     RTI_ATOMIC(RTI_UINT64) discovery_packets_received;
707     /*
708      * participant_discovery_bytes and participant_discovery_packets refer to
709      * DATA(P)s or participant bootstrap messages (i.e. messages sent/received
710      * by the anonymous participant writer/reader).
711      */
712     RTI_ATOMIC(RTI_UINT64) participant_discovery_bytes_sent;
713     RTI_ATOMIC(RTI_UINT64) participant_discovery_bytes_received;
714     RTI_ATOMIC(RTI_UINT64) participant_discovery_packets_sent;
715     RTI_ATOMIC(RTI_UINT64) participant_discovery_packets_received;
716     /*
717      * participant_config_bytes and participant_config_packets refer to
718      * participant configuration announcements sent reliably between
719      * participants.
720      */
721     RTI_ATOMIC(RTI_UINT64) participant_config_bytes_sent;
722     RTI_ATOMIC(RTI_UINT64) participant_config_bytes_received;
723     RTI_ATOMIC(RTI_UINT64) participant_config_packets_sent;
724     RTI_ATOMIC(RTI_UINT64) participant_config_packets_received;
725     RTI_ATOMIC(RTI_UINT64) automatic_liveliness_bytes_sent;
726     RTI_ATOMIC(RTI_UINT64) automatic_liveliness_bytes_received;
727     RTI_ATOMIC(RTI_UINT64) automatic_liveliness_packets_sent;
728     RTI_ATOMIC(RTI_UINT64) automatic_liveliness_packets_received;
729     RTI_ATOMIC(RTI_UINT64) manual_liveliness_bytes_sent;
730     RTI_ATOMIC(RTI_UINT64) manual_liveliness_bytes_received;
731     RTI_ATOMIC(RTI_UINT64) manual_liveliness_packets_sent;
732     RTI_ATOMIC(RTI_UINT64) manual_liveliness_packets_received;
733     RTI_ATOMIC(RTI_UINT64) endpoint_discovery_bytes_sent;
734     RTI_ATOMIC(RTI_UINT64) endpoint_discovery_bytes_received;
735     RTI_ATOMIC(RTI_UINT64) endpoint_discovery_packets_sent;
736     RTI_ATOMIC(RTI_UINT64) endpoint_discovery_packets_received;
737     /*
738      * Overhead statistics are for packets that use a builtin port (so are not
739      * user data), but are also not associated with participant or endpoint
740      * discovery
741      */
742     RTI_ATOMIC(RTI_UINT64) overhead_bytes_sent;
743     RTI_ATOMIC(RTI_UINT64) overhead_bytes_received;
744     RTI_ATOMIC(RTI_UINT64) overhead_packets_sent;
745     RTI_ATOMIC(RTI_UINT64) overhead_packets_received;
```

```
746     RTI_ATOMIC(RTI_UINT64) binding_ping_event_count;
747     RTI_ATOMIC(RTI_UINT64) binding_ping_packets_sent;
748     RTI_ATOMIC(RTI_UINT64) binding_ping_bytes_sent;
749     RTI_ATOMIC(RTI_UINT64) binding_ping_packets_received;
750     RTI_ATOMIC(RTI_UINT64) binding_ping_bytes_received;
751
752     /*
753      * Array of counters, one per DiffServ value. These counters will be
754      * incremented based on the received DiffServ values of the UDP packet.
755      *
756      * If received_priority_count[2] == 45 it means that we have received 45
757      * packets with DiffServ value 2.
758      *
759      * This field is populated only when gather_detailed_statistics is set
760      * to 1.
761      */
762     RTI_ATOMIC(RTI_UINT32)
763     received_priority_count[NDDS_TRANSPORT_UDP_DIFF_SERV_MAX_VALUE + 1];
764
765     /***********************
766      *   PRIVATE FIELDS
767      ***********************/
768     /*
769      * This field is used differently depending on whether or not we are using
770      * security. If we are not using security, this field will count the
771      * automatic liveliness messages sent and received. If we are using
772      * security, this field will count both automatic and manual liveliness
773      * messages sent and received.
774      */
775     RTI_ATOMIC(RTI_UINT64) _liveliness_bytes_sent;
776     RTI_ATOMIC(RTI_UINT64) _liveliness_bytes_received;
777     RTI_ATOMIC(RTI_UINT64) _liveliness_packets_sent;
778     RTI_ATOMIC(RTI_UINT64) _liveliness_packets_received;
779 };
780
781 /*i
782  * @brief Initializes NDDS_Transport_UDP_Statistics.
783  */
784 NDDS_Transport_DllExport void NDDS_Transport_UDP_Statistics_initialize(
785         struct NDDS_Transport_UDP_Statistics *stats);
786
787 /*i
788  * @brief Copies the content of an NDDS_Transport_UDP_Statistics object to
789  * another.
790  */
791 NDDS_Transport_DllExport void NDDS_Transport_UDP_copy_statistics(
792         struct NDDS_Transport_UDP_Statistics *dst,
793         const struct NDDS_Transport_UDP_Statistics *src);
794
795 /*i
796  * @brief Gets transport UDP statistics from self into statistics_out.
797  */
798 NDDS_Transport_DllExport void NDDS_Transport_UDP_get_statistics(
799         NDDS_Transport_Plugin *self,
800         struct NDDS_Transport_UDP_Statistics *statistics_out);
801
802 /*i
803  * @brief Gets the minimum and maximum transport priorities of the self's send
804  * resources.
805  */
806 NDDS_Transport_DllExport void NDDS_Transport_UDP_get_priority_range(
807         NDDS_Transport_Plugin *self,
808         RTI_INT32 *min_priority,
809         RTI_INT32 *max_priority);
810
811 /*i
812  * @brief Prints an address with a leading indent of "indent" spaces and a
813  * description string "%s:".
814  */
815 NDDS_Transport_DllExport void NDDS_Transport_UDP_Address_print(
816         const NDDS_Transport_Address_t *me,
817         const char *desc,
818         RTI_INT32 indent);
819
820 /*i
821  * @brief Gets bytes sent by an UDP transport in a time interval from before and
822  * after transport statistics.
823  */
824 NDDS_Transport_DllExport RTI_UINT64 NDDS_Transport_UDP_get_bytes_sent(
825         const struct NDDS_Transport_UDP_Statistics * const before,
826         const struct NDDS_Transport_UDP_Statistics * const after);
```

```
827
828 /*i
829  * @brief Gets packets sent by an UDP transport in a time interval from before
830  * and after transport statistics.
831  */
832 NDDS_Transport_DllExport RTI_UINT64 NDDS_Transport_UDP_get_packets_sent(
833         const struct NDDS_Transport_UDP_Statistics * const before,
834         const struct NDDS_Transport_UDP_Statistics * const after);
835
836 /*i
837  * @brief Gets bytes received by an UDP transport in a time interval from
838  * before and after transport statistics.
839  */
840 NDDS_Transport_DllExport RTI_UINT64 NDDS_Transport_UDP_get_bytes_received(
841         const struct NDDS_Transport_UDP_Statistics * const before,
842         const struct NDDS_Transport_UDP_Statistics * const after);
843
844 /*i
845  * @brief Gets packets received by an UDP transport in a time interval
846  * from before and after transport statistics.
847  */
848 NDDS_Transport_DllExport RTI_UINT64 NDDS_Transport_UDP_get_packets_received(
849         const struct NDDS_Transport_UDP_Statistics * const before,
850         const struct NDDS_Transport_UDP_Statistics * const after);
851
852 /* ================================================================= */
853 /*                           UDP WAN UUID                            */
854 /* ================================================================= */
855 /* Universal Unique IDentifier for a transport UDP_WAN instance. */
856 typedef struct {
857     RTI_UINT8 value[9];
858 } NDDS_Transport_UDP_WAN_UUID;
859
860 #define NDDS_Transport_UDP_WAN_UUID_Initializer \
861     {{0, 0, 0, 0, 0, 0, 0, 0, 0}}
862
863 extern NDDS_Transport_DllVariable const NDDS_Transport_UDP_WAN_UUID
864         NDDS_Transport_UDP_WAN_UUID_ZERO;
865
866 extern NDDS_Transport_DllExport RTIBool
867 NDDS_Transport_UDP_WAN_UUID_is_zero(NDDS_Transport_UDP_WAN_UUID *uuid);
868
879 extern NDDS_Transport_DllExport RTIBool NDDS_Transport_UDP_WAN_UUID_populate(
880         NDDS_Transport_UDP_WAN_UUID *self,
881         const struct RTIOsapiRtpsGuidPrefix *guidPrefix,
882         RTI_UINT32 hostAddress);
883
884 /* ================================================================= */
885 /*                 UDPv4 WAN Transport Adress Flag                   */
886 /* ================================================================= */
887
888 typedef RTI_UINT8 NDDS_Transport_UDP_WAN_AddressFlag;
889
890
891 #define NDDS_TRANSPORT_UDP_WAN_ADDRESS_FLAG_NONE (0)
892
893 /*
894  * The U flag indicates if the locator contains a UUID. While this identifier by
895  * itself cannot be directly used to reach the transport instance, the UUID can
896  * be mapped to a public IP  address by  other Connext DDS applications and
897  * services. Also, a locator can have both, the U flag, and the P flag, enabled
898  * simultaneously. Locators with the U flag set are called UUID locators.
899  */
900 #define NDDS_TRANSPORT_UDP_WAN_ADDRESS_FLAG_U (0x01U « 0)
901 /*
902  * The P flag indicates that the locator contains a globally public IP address
903  * and public port where a transport instance can be reached. public_ip_address
904  * contains the public IP address and public_port contains the public UDP port.
905  * The port is always used to receive data and if the B flag is set it is also
906  * used to send data. Locators with the P flag set are called PUBLIC locators.
907  */
908 #define NDDS_TRANSPORT_UDP_WAN_ADDRESS_FLAG_P (0x01U « 1)
909 /*
910  * The B flag indicates if the locator is unidirectional or bidirectional.
911  * Bidirectional locators can send/receive RTPS traffic. Unidirectional locators
912  * can only receive RTPS traffic.
913  */
914 #define NDDS_TRANSPORT_UDP_WAN_ADDRESS_FLAG_B (0x01U « 2)
915 /*
916  * The R flag indicates if the locator is a relay locator provided by a Relay
917  * Service. Locators with the R flag set are called RELAY locators.
```

```
 */
#define NDDS_TRANSPORT_UDP_WAN_ADDRESS_FLAG_R (0x01U << 3)

/* ================================================================= */
/*                 UDPv4 WAN Transport Address Public Port           */
/* ================================================================= */

typedef struct {
    RTI_UINT8 network_ordered_value[2];
} NDDS_Transport_UDP_WAN_NetworkPort;

#define NDDS_TRANSPORT_UDP_WAN_NETWORK_PORT_INVALID {{0}}

/* ================================================================= */
/*                 UDPv4 WAN Transport Address Public Address        */
/* ================================================================= */

typedef struct {
    RTI_UINT8 network_ordered_value[4];
} NDDS_Transport_UDP_WAN_NetworkIpv4Address;

#define NDDS_TRANSPORT_UDP_WAN_NETWORK_IP_ADDRESS_INVALID {{0}}

/* ================================================================= */
/*                 UDPv4 WAN Transport Address                       */
/* ================================================================= */
/*
 * UDPv4_WAN locators use the following NDDS_Transport_Address_t mapping:
 *
 * +-------+-------+-------+-------+
 * | Flags |                       |
 * +-------+                       +
 * |       DDS_Octet UUID[9]       |
 * |               +-------+-------+
 * |               |  public_port  |
 * +-------+-------+-------+-------+
 * | DDS_Octet public_ip_address[4]|
 * +-------+-------+-------+-------+
 */

typedef struct {
    NDDS_Transport_UDP_WAN_AddressFlag _flags;
    NDDS_Transport_UDP_WAN_UUID _uuid;
    NDDS_Transport_UDP_WAN_NetworkPort _publicPort;
    NDDS_Transport_UDP_WAN_NetworkIpv4Address _publicAddress;
} NDDS_Transport_UDPv4_WAN_Address;

extern NDDS_Transport_DllExport NDDS_Transport_UDP_Port
NDDS_Transport_UDPv4_WAN_Address_get_public_port(
        const NDDS_Transport_UDPv4_WAN_Address *self);

extern NDDS_Transport_DllExport NDDS_Transport_UDP_Port
NDDS_Transport_UDPv4_WAN_Address_get_public_port_for_rtps_port(
        const NDDS_Transport_UDPv4_WAN_Address *self,
        NDDS_Transport_Port_t rtps_port,
        RTI_INT32 udp_port_offset);

extern NDDS_Transport_DllExport void
NDDS_Transport_UDPv4_WAN_Address_set_public_port(
        NDDS_Transport_UDPv4_WAN_Address *self,
        NDDS_Transport_UDP_Port public_port);

extern NDDS_Transport_DllExport RTI_UINT32
NDDS_Transport_UDPv4_WAN_Address_get_public_address(
        const NDDS_Transport_UDPv4_WAN_Address *self);

extern NDDS_Transport_DllExport void
NDDS_Transport_UDPv4_WAN_Address_set_public_address(
        NDDS_Transport_UDPv4_WAN_Address *self,
        RTI_UINT32 public_address);

extern NDDS_Transport_DllExport void NDDS_Transport_UDP_combine_addresses(
         const NDDS_Transport_Plugin *self,
         NDDS_Transport_Address_t *address_out,
         const NDDS_Transport_Address_t *first_address_in,
         const NDDS_Transport_Address_t *second_address_in);

/*i
 * @brief Gets the public address mapping in the provided address for the provided
 * (UUID, RTPS port) tuple.
 *
```

```
1013  * @param self \b In. Plugin, must be WAN.
1014  * @param address \b Out. Address to populate with the binding ping mapping.
1015  * @param rtpsPort \b In. Binding ping RTPS port.
1016  * @param uuid \b In. Binding ping UUID.
1017  *
1018  * @return RTI_TRUE if success, RTI_FALSE if error or if the binding ping
1019  * mapping does not exist.
1020  */
1021 extern NDDS_Transport_DllExport RTIBool
1022 NDDS_Transport_UDP_get_binding_ping_mapping(
1023         NDDS_Transport_Plugin *self,
1024         NDDS_Transport_UDPv4_WAN_Address *address,
1025         NDDS_Transport_Port_t rtpsPort,
1026         const NDDS_Transport_UDP_WAN_UUID *uuid);
1027
1028 #ifdef __cplusplus
1029 } /* extern "C" */
1030 #endif
1031
1032 #include "transport/transport_udp_impl.h"
1033
1034 #endif /* transport_udp_h */
```

10.7 transport_udpv4.h

```
1  /*
2   * (c) Copyright, Real-Time Innovations, 2005-2024.
3   * All rights reserved.
4   *
5   * No duplications, whole or partial, manual or electronic, may be made
6   * without express written permission.  Any such copies, or
7   * revisions thereof, must display this notice unaltered.
8   * This code contains trade secrets of Real-Time Innovations, Inc.
9   */
10
11 #ifndef transport_udpv4_h
12 #define transport_udpv4_h
13
14 #include "transport/transport_udp.h"
15
16 #ifdef __cplusplus
17 extern "C" {
18 #endif
19
20 #define NDDS_Transport_UDPv4 NDDS_Transport_UDP
21
22 #define NDDS_Transport_UDPv4_InterfaceListener \
23   NDDS_Transport_UDP_InterfaceListener
24
25 /* ================================================================= */
26 /*                 Component: UDP/IPv4 Transport Plugin              */
27 /* ================================================================= */
28
29 /*e \dref_UDPv4TransportBuiltinGroupDocs
30  */
31
32 /*i \file
33  * \ingroup NDDS_Transport_UDPv4_Plugin
34  * \brief UDPv4 Transport Plugin interfaces and definitions.
35  */
36
37 /* ================================================================= */
38 /*                 UDPv4 Transport Properties                        */
39 /* ================================================================= */
40
41 #ifdef DOXYGEN_DOCUMENTATION_ONLY
42 /*e \dref_UDPv4Transport_Property_t
43  */
44 struct NDDS_Transport_UDPv4_Property_t {
45     /*e \dref_UDPv4Transport_Property_t_parent */
46     struct NDDS_Transport_Property_t parent;
47
48     /*e \dref_UDPv4Transport_Property_t_send_socket_buffer_size */
49     RTI_INT32 send_socket_buffer_size;
50
51     /*e \dref_UDPv4Transport_Property_t_recv_socket_buffer_size */
52     RTI_INT32 recv_socket_buffer_size;
```

```
53
54      /*e \dref_UDPv4Transport_Property_t_unicast_enabled */
55      RTI_INT32 unicast_enabled;
56
57      /*e \dref_UDPv4Transport_Property_t_multicast_enabled */
58      RTI_INT32 multicast_enabled;
59
60      /*e \dref_UDPv4Transport_Property_t_multicast_ttl */
61      RTI_INT32 multicast_ttl;
62
63      /*e \dref_UDPv4Transport_Property_t_multicast_loopback_disabled */
64      RTI_INT32 multicast_loopback_disabled;
65
66      /*e \dref_UDPv4Transport_Property_t_ignore_loopback_interface */
67      RTI_INT32 ignore_loopback_interface;
68
69      /*e \dref_UDPv4Transport_Property_t_ignore_nonup_interfaces */
70      RTI_INT32 ignore_nonup_interfaces;
71
72      /*e \dref_UDPv4Transport_Property_t_ignore_nonrunning_interfaces */
73      RTI_INT32 ignore_nonrunning_interfaces;
74
75      /*e \dref_UDPv4Transport_Property_t_no_zero_copy */
76      RTI_INT32 no_zero_copy;
77
78      /*e \dref_UDPv4Transport_Property_t_send_blocking */
79      RTI_INT32 send_blocking;
80
81      /*e \dref_UDPv4Transport_Property_t_use_checksum */
82      RTI_INT32 use_checksum;
83
84      /*e \dref_UDPv4Transport_Property_t_transport_priority_mask */
85      RTI_UINT32 transport_priority_mask;
86
87      /*e \dref_UDPv4Transport_Property_t_transport_priority_mapping_low */
88      RTI_INT32 transport_priority_mapping_low;
89
90      /*e \dref_UDPv4Transport_Property_t_transport_priority_mapping_high */
91      RTI_INT32 transport_priority_mapping_high;
92
93      /*e \dref_UDPv4Transport_Property_t_send_ping */
94      RTI_INT32 send_ping;
95
96      /*e \dref_UDPv4Transport_Property_t_force_interface_poll_detection */
97      RTI_INT32 force_interface_poll_detection;
98
99      /*e \dref_UDPv4Transport_Property_t_interface_poll_period */
100      RTI_UINT32 interface_poll_period;
101
102      /*e \dref_UDPv4Transport_Property_t_reuse_multicast_receive_resource */
103      RTI_INT32 reuse_multicast_receive_resource;
104
105      /*e \dref_UDPv4Transport_Property_t_protocol_overhead_max */
106      RTI_INT32 protocol_overhead_max;
107
108      /*e \dref_UDPv4Transport_Property_t_disable_interface_tracking */
109      RTI_INT32 disable_interface_tracking;
110
111      /*e \dref_UDPv4Transport_Property_t_join_multicast_group_timeout */
112      RTI_UINT32 join_multicast_group_timeout;
113
114      /*e \dref_UDPv4Transport_Property_t_public_address */
115      char *public_address;
116
117      /*e \dref_UDPv4Transport_Property_t_port_offset */
118      RTI_INT32 port_offset;
119  };
120  #endif
121
122  /* ================================================================= */
123  /*                 Component: UDP/IPv4  WAN Transport Plugin         */
124  /* ================================================================= */
125
126  /*e \dref_UDPv4WanTransportBuiltinGroupDocs */
127
128  /*i \file
129   * \ingroup NDDS_Transport_UDPv4_WAN_Plugin
130   * \brief UDPv4 WAN Transport Plugin interfaces and definitions.
131   */
132
133  /* ================================================================= */
```

```
134 /*                 UDPv4 WAN Transport Properties                     */
135 /* ================================================================== */
136
137 #ifdef DOXYGEN_DOCUMENTATION_ONLY
138 /*e \dref_UDPv4_WAN_Transport_Property_t
139  */
140 struct NDDS_Transport_UDPv4_WAN_Property_t {
141     /*e \dref_UDPv4_WAN_Transport_Property_t_parent */
142     struct NDDS_Transport_Property_t parent;
143
144     /*e \dref_UDPv4_WAN_Transport_Property_t_send_socket_buffer_size */
145     RTI_INT32 send_socket_buffer_size;
146
147     /*e \dref_UDPv4_WAN_Transport_Property_t_recv_socket_buffer_size */
148     RTI_INT32 recv_socket_buffer_size;
149
150     /*e \dref_UDPv4_WAN_Transport_Property_t_ignore_loopback_interface */
151     RTI_INT32 ignore_loopback_interface;
152
153     /*e \dref_UDPv4_WAN_Transport_Property_t_ignore_nonup_interfaces */
154     RTI_INT32 ignore_nonup_interfaces;
155
156     /*e \dref_UDPv4_WAN_Transport_Property_t_ignore_nonrunning_interfaces */
157     RTI_INT32 ignore_nonrunning_interfaces;
158
159     /*e \dref_UDPv4_WAN_Transport_Property_t_no_zero_copy */
160     RTI_INT32 no_zero_copy;
161
162     /*e \dref_UDPv4_WAN_Transport_Property_t_send_blocking */
163     RTI_INT32 send_blocking;
164
165     /*e \dref_UDPv4_WAN_Transport_Property_t_use_checksum */
166     RTI_INT32 use_checksum;
167
168     /*e \dref_UDPv4_WAN_Transport_Property_t_transport_priority_mask */
169     RTI_UINT32 transport_priority_mask;
170
171     /*e \dref_UDPv4_WAN_Transport_Property_t_transport_priority_mapping_low */
172     RTI_INT32 transport_priority_mapping_low;
173
174     /*e \dref_UDPv4_WAN_Transport_Property_t_transport_priority_mapping_high */
175     RTI_INT32 transport_priority_mapping_high;
176
177     /*e \dref_UDPv4_WAN_Transport_Property_t_send_ping */
178     RTI_INT32 send_ping;
179
180     /*e \dref_UDPv4_WAN_Transport_Property_t_force_interface_poll_detection */
181     RTI_INT32 force_interface_poll_detection;
182
183     /*e \dref_UDPv4_WAN_Transport_Property_t_interface_poll_period */
184     RTI_UINT32 interface_poll_period;
185
186     /*e \dref_UDPv4_WAN_Transport_Property_t_protocol_overhead_max */
187     RTI_INT32 protocol_overhead_max;
188
189     /*e \dref_UDPv4_WAN_Transport_Property_t_disable_interface_tracking */
190     RTI_INT32 disable_interface_tracking;
191
192     /*e \dref_UDPv4_WAN_Transport_Property_t_public_address */
193     char *public_address;
194
195     /*e \dref_UDPv4_WAN_Transport_Property_t_comm_ports_list */
196     struct NDDS_Transport_UDP_WAN_CommPortsMappingInfo *comm_ports_list;
197
198     /*e \dref_UDPv4_WAN_Transport_Property_t_comm_ports_list_length */
199     RTI_INT32 comm_ports_list_length;
200
201     /*e \dref_UDPv4_WAN_Transport_Property_t_port_offset */
202     RTI_INT32 port_offset;
203
204     /*e \dref_UDPv4_WAN_Transport_Property_t_binding_ping_period */
205     RTI_UINT32 binding_ping_period;
206 };
207 #endif
208
209 #define NDDS_Transport_UDPv4_Property_t NDDS_Transport_UDP_Property_t
210 #define NDDS_Transport_UDPv4_WAN_Property_t NDDS_Transport_UDP_Property_t
211
212 /*i \ingroup NDDS_Transport_UDPv4_Plugin
213  * \brief UDPv4 plugin class string name
214  */
```

```
215 #define NDDS_TRANSPORT_UDPV4_CLASS_NAME "udpv4"
216
217 /*i \ingroup NDDS_Transport_UDPv4_Plugin
218  * \brief UDPv4 plugin class string name
219  */
220 #define NDDS_TRANSPORT_UDPV4_WAN_CLASS_NAME "udpv4_wan"
221
222 /*i \ingroup NDDS_Transport_UDPv4_Plugin
223  * \brief UDPv4 builtin plugin alias
224  */
225 #define NDDS_TRANSPORT_BUILTIN_UDPV4_ALIAS "builtin.udpv4"
226
227 /*i \ingroup NDDS_Transport_UDPv4_Plugin
228  * \brief UDPv4 builtin plugin alias
229  */
230 #define NDDS_TRANSPORT_BUILTIN_UDPV4_WAN_ALIAS "builtin.udpv4_wan"
231
232 /*e \ingroup NDDS_Transport_UDPv4_Plugin
233  * \brief Default value of NDDS_Transport_Property_t#address_bit_count
234  */
235 #define NDDS_TRANSPORT_UDPV4_ADDRESS_BIT_COUNT (32)
236
237 /*e \ingroup NDDS_Transport_UDPv4_Plugin
238  * \brief Default value of NDDS_Transport_Property_t#address_bit_count for
239  * UDPv4 asymmetric transport
240  */
241 #define NDDS_TRANSPORT_UDPV4_WAN_ADDRESS_BIT_COUNT (128)
242
243 /*e \ingroup NDDS_Transport_UDPv4_Plugin
244  * \brief Default value of NDDS_Transport_Property_t#properties_bitmap.
245  */
246 #define NDDS_TRANSPORT_UDPV4_PROPERTIES_BITMAP_DEFAULT \
247     NDDS_TRANSPORT_UDP_PROPERTIES_BITMAP_DEFAULT
248
249 /*e \ingroup NDDS_Transport_UDPv4_Plugin
250  * \brief Default value of
251  * NDDS_Transport_Property_t#gather_send_buffer_count_max.
252  *
253  * This is also the maximum value that can be used when instantiating the UDP
254  * transport.
255  *
256  * 16 is sufficient for \ndds, but more may improve discovery and reliable
257  * performance.
258  * Porting note: find out what the maximum gather buffer count is on your OS!
259  */
260 #define NDDS_TRANSPORT_UDPV4_GATHER_SEND_BUFFER_COUNT_MAX_DEFAULT \
261     NDDS_TRANSPORT_UDP_GATHER_SEND_BUFFER_COUNT_MAX_DEFAULT
262
263 /*e \dref_UDPv4Transport_SOCKET_BUFFER_SIZE_OS_MAX
264  */
265 #define NDDS_TRANSPORT_UDPV4_SOCKET_BUFFER_SIZE_OS_MAX \
266     NDDS_TRANSPORT_UDP_SOCKET_BUFFER_SIZE_OS_MAX
267
268 /*e \dref_UDPv4Transport_SOCKET_BUFFER_SIZE_OS_DEFAULT
269  */
270 #define NDDS_TRANSPORT_UDPV4_SOCKET_BUFFER_SIZE_OS_DEFAULT \
271     NDDS_TRANSPORT_UDP_SOCKET_BUFFER_SIZE_OS_DEFAULT
272
273 /*e \dref_UDPv4Transport_SEND_SOCKET_BUFFER_SIZE_DEFAULT
274  */
275 #define NDDS_TRANSPORT_UDPV4_SEND_SOCKET_BUFFER_SIZE_DEFAULT \
276     NDDS_TRANSPORT_UDP_SEND_SOCKET_BUFFER_SIZE_DEFAULT
277
278 /*e \dref_UDPv4Transport_RECV_SOCKET_BUFFER_SIZE_DEFAULT
279  */
280 #define NDDS_TRANSPORT_UDPV4_RECV_SOCKET_BUFFER_SIZE_DEFAULT \
281     NDDS_TRANSPORT_UDP_RECV_SOCKET_BUFFER_SIZE_DEFAULT
282
283 /*e \ingroup NDDS_Transport_UDPv4_Plugin
284  * \brief Maximum value of NDDS_Transport_Property_t#message_size_max.
285  */
286 #define NDDS_TRANSPORT_UDPV4_PAYLOAD_SIZE_MAX (65507)
287
288 /*e \ingroup NDDS_Transport_UDPv4_Plugin
289  * \brief Maximum value of NDDS_Transport_Property_t#message_size_max.
290  */
291 #if defined(RTI_INTY)
292   #define NDDS_TRANSPORT_UDPV4_MESSAGE_SIZE_MAX_DEFAULT (9216)
293 #else
294   #define NDDS_TRANSPORT_UDPV4_MESSAGE_SIZE_MAX_DEFAULT \
295     NDDS_TRANSPORT_UDPV4_PAYLOAD_SIZE_MAX
```

```
296 #endif
297
298 /*e \ingroup NDDS_Transport_UDPv4_WAN_Plugin
299  * \brief Maximum value of NDDS_Transport_Property_t#message_size_max.
300  */
301 #define NDDS_TRANSPORT_UDPV4_WAN_MESSAGE_SIZE_MAX_DEFAULT (1400)
302
303 #define NDDS_TRANSPORT_UDPV4_DATAGRAM_LENGTH_MAX \
304   NDDS_TRANSPORT_UDP_DATAGRAM_LENGTH_MAX
305
306 /*e \ingroup NDDS_Transport_UDPv4_Plugin
307  * \brief Default value of NDDS_Transport_UDPv4_Property_t#multicast_ttl.
308  */
309 #define NDDS_TRANSPORT_UDPV4_MULTICAST_TTL_DEFAULT \
310   NDDS_TRANSPORT_UDP_MULTICAST_TTL_DEFAULT
311
312
313 #define NDDS_TRANSPORT_UDPV4_USE_MULTICAST_DEFAULT \
314   NDDS_TRANSPORT_UDP_USE_MULTICAST_DEFAULT
315
316
317 /*e \dref_UDPv4Transport_IGNORE_LOOPBACK_DISABLED
318  */
319 #define NDDS_TRANSPORT_UDPV4_IGNORE_LOOPBACK_DISABLED \
320   NDDS_TRANSPORT_UDP_IGNORE_LOOPBACK_DISABLED
321
322 /*e \dref_UDPv4Transport_IGNORE_LOOPBACK_ENABLED
323  */
324 #define NDDS_TRANSPORT_UDPV4_IGNORE_LOOPBACK_ENABLED \
325   NDDS_TRANSPORT_UDP_IGNORE_LOOPBACK_ENABLED
326
327 /*e \dref_UDPv4Transport_IGNORE_LOOPBACK_AUTO
328  */
329 #define NDDS_TRANSPORT_UDPV4_IGNORE_LOOPBACK_AUTO \
330   NDDS_TRANSPORT_UDP_IGNORE_LOOPBACK_AUTO
331
332
333 /*e \dref_UDPv4Transport_USE_CHECKSUM_DISABLED
334  */
335 #define NDDS_TRANSPORT_UDPV4_USE_CHECKSUM_DISABLED (0)
336
337 /*e \dref_UDPv4Transport_USE_CHECKSUM_ENABLED
338  */
339 #define NDDS_TRANSPORT_UDPV4_USE_CHECKSUM_ENABLED (1)
340
341
342 /*e \dref_UDPv4Transport_BLOCKING_NEVER
343  */
344 #define NDDS_TRANSPORT_UDPV4_BLOCKING_NEVER NDDS_TRANSPORT_UDP_BLOCKING_NEVER
345
346 /*e \dref_UDPv4Transport_BLOCKING_ALWAYS
347  */
348 #define NDDS_TRANSPORT_UDPV4_BLOCKING_ALWAYS NDDS_TRANSPORT_UDP_BLOCKING_ALWAYS
349 #define NDDS_TRANSPORT_UDPV4_BLOCKING_UNICAST_ONLY \
350   NDDS_TRANSPORT_UDP_BLOCKING_UNICAST_ONLY
351
352 /*e \dref_UDPv4Transport_BLOCKING_DEFAULT
353  */
354 #define NDDS_TRANSPORT_UDPV4_BLOCKING_DEFAULT \
355   NDDS_TRANSPORT_UDP_BLOCKING_DEFAULT
356
357 /*e \ingroup NDDS_Transport_UDPv4_Plugin
358  * \brief Use this to initialize a NDDS_Transport_UDPv4_Property_t structure.
359  */
360 #define NDDS_TRANSPORT_UDPV4_PROPERTY_DEFAULT \
361     { \
362         { \
363             NDDS_TRANSPORT_CLASSID_UDPv4, \
364             NDDS_TRANSPORT_UDPV4_ADDRESS_BIT_COUNT, \
365             NDDS_TRANSPORT_UDPV4_PROPERTIES_BITMAP_DEFAULT, \
366             NDDS_TRANSPORT_UDPV4_GATHER_SEND_BUFFER_COUNT_MAX_DEFAULT, \
367             NDDS_TRANSPORT_UDPV4_MESSAGE_SIZE_MAX_DEFAULT, \
368             NULL, \
369             0, /* allow_interfaces_list */ \
370             NULL, \
371             0, /* deny_interfaces_list */ \
372             NULL, \
373             0, /* allow_multicast_interfaces_list */ \
374             NULL, \
375             0,                          /* deny_multicast_interfaces_list */ \
376             NDDS_TRANSPORT_UUID_UNKNOWN, /* transport_uuid */ \
```

```
377             NULL,                       /* domain_participant_ptr */ \
378             NULL,                       /* domain_participant_forwarder */ \
379             NULL,                       /* thread_name_prefix */ \
380             NDDS_TRANSPORT_LENGTH_UNLIMITED, /* max_interface_count */ \
381         }, \
382                 NDDS_TRANSPORT_UDPV4_SEND_SOCKET_BUFFER_SIZE_DEFAULT, \
383                 NDDS_TRANSPORT_UDPV4_RECV_SOCKET_BUFFER_SIZE_DEFAULT, \
384                 1,                                          /* unicast_enabled \
385                                                              */ \
386                 NDDS_TRANSPORT_UDPV4_USE_MULTICAST_DEFAULT, /* multicast_enabled \
387                                                              */ \
388                 NDDS_TRANSPORT_UDPV4_MULTICAST_TTL_DEFAULT, 0, /* multicast \
389                                                                   loopback \
390                                                                   enabled */ \
391                 NDDS_TRANSPORT_UDPV4_IGNORE_LOOPBACK_AUTO, /* (auto-)ignore \
392                                                               loopback */ \
393                 1, /* ignore_nonup_interfaces */ \
394                 1, /* ignore_nonrunning_interfaces */ \
395                 0, /* no_zero_copy */ \
396                 NDDS_TRANSPORT_UDPV4_BLOCKING_DEFAULT, 0,  /* enable_v4mapped \
397                                                               */ \
398                 NDDS_TRANSPORT_UDPV4_USE_CHECKSUM_ENABLED, /* use_checksum */ \
399                 0, 0, 0xff /* no mapping to IP_TOS by default */, \
400                 1,    /* send_ping \
401                        */ \
402                 0,    /* don't force polling interface tracker by default */ \
403                 500,  /* 500 millisecs is the default polling period*/ \
404                 1,    /* reuse multicast receive resource */ \
405                 28,   /* protocol_overhead_max (UDP(8)+IP(20)) */ \
406                 0,    /* disable interface tracking */ \
407                 5000, /* join_multicast_group_timeout */ \
408                 NULL, /* public_address */ \
409                 NULL, 0,   /* comm_ports_list */ \
410                 NULL,      /* wan_listener */ \
411                 0,         /* port_offset */ \
412                 0,         /* binding_ping_period */ \
413                 0,         /* binding_ping_mapping_expiration */ \
414                 NULL,      /* plugin_enabled_function_ptr */ \
415                 RTI_FALSE, /* gather_detailed_statistics */ \
416                 RTI_FALSE, /* drop_samples_by_guid_prefix */ \
417                 RTI_OSAPI_RTPS_GUID_PREFIX_UNKNOWN /* guid_prefix_to_drop */ \
418     }
419
420 /*e \ingroup NDDS_Transport_UDPv4_WAN_Plugin
421  * \brief Use this to initialize a NDDS_Transport_UDPv4_WAN_Property_t
422  * structure.
423  */
424 #define NDDS_TRANSPORT_UDPV4_WAN_PROPERTY_DEFAULT \
425     { \
426         { \
427             NDDS_TRANSPORT_CLASSID_UDPv4_WAN, \
428             NDDS_TRANSPORT_UDPV4_WAN_ADDRESS_BIT_COUNT, \
429             NDDS_TRANSPORT_UDPV4_PROPERTIES_BITMAP_DEFAULT, \
430             NDDS_TRANSPORT_UDPV4_GATHER_SEND_BUFFER_COUNT_MAX_DEFAULT, \
431             NDDS_TRANSPORT_UDPV4_WAN_MESSAGE_SIZE_MAX_DEFAULT, \
432             NULL, \
433             0, /* allow_interfaces_list */ \
434             NULL, \
435             0, /* deny_interfaces_list */ \
436             NULL, \
437             0, /* allow_multicast_interfaces_list */ \
438             NULL, \
439             0,                          /* deny_multicast_interfaces_list */ \
440             NDDS_TRANSPORT_UUID_UNKNOWN, /* transport_uuid */ \
441             NULL,                       /* domain_participant_ptr */ \
442             NULL,                       /* domain_participant_forwarder */ \
443             NULL,                       /* thread_name_prefix */ \
444             NDDS_TRANSPORT_LENGTH_UNLIMITED, /* max_interface_count */ \
445         }, \
446                 NDDS_TRANSPORT_UDPV4_SEND_SOCKET_BUFFER_SIZE_DEFAULT, \
447                 NDDS_TRANSPORT_UDPV4_RECV_SOCKET_BUFFER_SIZE_DEFAULT, \
448                 1, /* unicast_enabled \
449                     */ \
450                 0, /* multicast_enabled */ \
451                 NDDS_TRANSPORT_UDPV4_MULTICAST_TTL_DEFAULT, 0, /* multicast \
452                                                                   loopback \
453                                                                   enabled */ \
454                 NDDS_TRANSPORT_UDPV4_IGNORE_LOOPBACK_AUTO, /* (auto-)ignore \
455                                                               loopback */ \
456                 1, /* ignore_nonup_interfaces */ \
457                 1, /* ignore_nonrunning_interfaces */ \
```

```
458                 0, /* no_zero_copy */ \
459                 NDDS_TRANSPORT_UDPV4_BLOCKING_DEFAULT, 0,  /* enable_v4mapped \
460                                                            */ \
461                 NDDS_TRANSPORT_UDPV4_USE_CHECKSUM_ENABLED, /* use_checksum */ \
462                 0, 0, 0xff /* no mapping to IP_TOS by default */, \
463                 1,    /* send_ping \
464                          */ \
465                 0,    /* don't force polling interface tracker by default */ \
466                 500,  /* 500 millisecs is the default polling period*/ \
467                 1,    /* reuse multicast receive resource */ \
468                 28,   /* protocol_overhead_max (UDP(8)+IP(20)) */ \
469                 0,    /* disable interface tracking */ \
470                 5000, /* join_multicast_group_timeout */ \
471                 NULL, /* public_address */ \
472                 NULL, 0,   /* comm_ports_list */ \
473                 NULL,      /* wan_listener */ \
474                 125,       /* port_offset */ \
475                 1000,      /* binding_ping_period */ \
476                 0,         /* binding_ping_mapping_expiration */ \
477                 NULL,      /* plugin_enabled_function_ptr */ \
478                 RTI_FALSE, /* gather_detailed_statistics */ \
479                 RTI_FALSE, /* drop_samples_by_guid_prefix */ \
480                 RTI_OSAPI_RTPS_GUID_PREFIX_UNKNOWN /* guid_prefix_to_drop */ \
481     }
482
483 /* ================================================================= */
484 /*                   InterfaceListener Class                         */
485 /* ================================================================= */
486
487 /*i
488  * \defgroup NDDS_Transport_UDP_InterfaceListenerClass
489  * NDDS_Transport_UDP_InterfaceListener \ingroup NDDS_Transport_UDP_Plugin
490  * \brief Interface for IPv4 interface acceptance/rejection
491  *
492  * @mtsafety See NDDS_Transport_UDP_public UML diagram
493  *
494  * @see NDDS_Transport_UDPv4_newI
495  */
496 #define NDDS_Transport_UDPv4_InterfaceListener \
497   NDDS_Transport_UDP_InterfaceListener
498
499 /*i \ingroup NDDS_Transport_UDPv4_InterfaceListenerClass
500  * \brief A user-provided function to be used to filter out particular
501  * network interfaces to be used by the Transport Plugin.
502  *
503  * The user may implement this function and pass it to an
504  * instance of an UDPv4 Transport Plugin to reject or accept network
505  * interfaces that the plugin offers as ones to use to receive
506  * messages.
507  *
508  * When \ndds contacts other applications, it will provide a list of
509  * interfaces that other applications can use to send messages (on a
510  * plugin by plugin basis).  The addresses of the interfaces are used
511  * as the destination addresses of messages that are sent.  If a plugin
512  * rejects one of the available network interfaces, then that interface
513  * will not be offered as one to which other plugins should send
514  * messages.
515  *
516  * For interfaces that can be used by the plugin, this function also
517  * enables the user to select which interfaces \ndds is allowed to use
518  * in sending multicast messages.  The set of interfaces enabled for
519  * multicast sending is a subset of the interfaces that can be used for
520  * receiving.  That is, an interface cannot be enabled to send
521  * multicast messages but disabled for receiving unicast messages.
522  *
523  * Only IP network interfaces that are "UP" (enabled by the OS and
524  * ready to use) will be passed to this filter function.  The ones that
525  * are "DOWN" will not be used by definition.
526  *
527  * @param plugin \st_in The instance of the plugin for which the filter
528  * is being called.
529  *
530  * @param multicast_enabled_out \st_out The filter should set whether
531  * or not the plugin is allowed to use this interface for
532  * sending/receiving multicast messages.
533  *
534  * @param interface_in \st_in An interface offered by the plugin as a
535  * candidate for receive messages.
536  *
537  * @return 1 to use the interface, 0 to reject using the interface.
538  *         \c multicast_enabled_out will indicate whether it is also OK
```

```
539  *         to use the interface to send multicast messages.
540  */
541 #define NDDS_Transport_UDPv4_Interface_Filter_Fcn \
542   NDDS_Transport_UDP_Interface_Filter_Fcn
543
544 /* ================================================================= */
545 /*                    UDPv4/IPv4 InterfaceListener                   */
546 /* ================================================================= */
547
548
549 /*i \ingroup NDDS_Transport_UDPv4_InterfaceListenerClass
550  * \brief Default implementation of the Interface listener.
551  *
552  * The default interface listener applies the interface selection preferences
553  * specified in the NDDS_Transport_UDP_Property_t#parent:
554  * - NDDS_Transport_Property_t#allow_interfaces_list,
555  * - NDDS_Transport_Property_t#deny_interfaces_list,
556  * - NDDS_Transport_Property_t#allow_multicast_interfaces_list,
557  * - NDDS_Transport_Property_t#deny_multicast_interfaces_list
558  *
559  * The filter assumes that the strings passed in the address filter
560  * are in UDPv4 dot notation, possibly containing wildcards. For example:
561  *        10.10.30.232
562  *        10.10.*.*
563  *        192.168.1.*
564  * etc.
565  *
566  * @see NDDS_Transport_UDP_Interface_Filter_Fcn
567  */
568 extern NDDS_Transport_DllExport RTI_INT32
569 NDDS_Transport_UDPv4_InterfaceListener_onInterface(
570         struct NDDS_Transport_UDPv4_InterfaceListener *self,
571         struct NDDS_Transport_UDPv4 *plugin,
572         RTI_INT32 *multicast_enabled_out,
573         NDDS_Transport_Interface_t *interface_in,
574         const char *interface_name_in);
575
576 /*i \ingroup NDDS_Transport_UDPv4_InterfaceListenerClass
577  * \brief Default interface listener.
578  */
579 extern NDDS_Transport_DllVariable struct NDDS_Transport_UDP_InterfaceListener
580         NDDS_Transport_UDPv4_g_interfaceListener;
581
582 /* ================================================================= */
583 /*                     Socket Factory Class                          */
584 /* ================================================================= */
585 /*i
586  * \defgroup NDDS_Transport_UDP_SocketFactoryClass
587  * NDDS_Transport_UDP_SocketFactory \ingroup NDDS_Transport_UDPv4_Plugin \brief
588  * Interface for IPv4 socket creation and destruction.
589  *
590  * @mtsafety See NDDS_Transport_UDPv4_public UML diagram
591  *
592  * @see NDDS_Transport_UDPv4_new
593  */
594
595 #define NDDS_Transport_UDPv4_SocketFactory_Send_Socket_Create_Fcn \
596   NDDS_Transport_UDP_SocketFactory_Send_Socket_Create_Fcn
597
598
599 #define NDDS_Transport_UDPv4_SocketFactory_Receive_Socket_Create_Fcn \
600   NDDS_Transport_UDP_SocketFactory_Receive_Socket_Create_Fcn
601
602
603 #define NDDS_Transport_UDPv4_SocketFactory_Socket_Destroy_Fcn \
604   NDDS_Transport_UDP_SocketFactory_Socket_Destroy_Fcn
605
606
607 #define NDDS_Transport_UDPv4_SocketFactory NDDS_Transport_UDP_SocketFactory
608
609
610 #define NDDS_Transport_UDPv4_SocketFactory_is_valid \
611   NDDS_Transport_UDP_SocketFactory_is_valid
612
613 /* ================================================================= */
614 /*                  IP Socket Factory Class                          */
615 /* ================================================================= */
616
617 /*i \ingroup NDDS_Transport_UDP_SocketFactoryClass
618  * \brief Default implementation of the socket creation factory method
619  * for UDPv4 transport.
```

```
620  *
621  * @post Sending socket is not bound.
622  *
623  * @see NDDS_Transport_UDP_SocketFactory_Send_Socket_Create_Fcn
624  */
625 extern NDDS_Transport_DllExport RTI_INT32
626 NDDS_Transport_UDPv4_SocketFactory_create_send_socket(
627         struct NDDS_Transport_UDPv4_SocketFactory *me,
628         NDDS_Transport_Port_t *port_out,
629         const struct NDDS_Transport_UDPv4 *ip_in,
630         const NDDS_Transport_Address_t *interface_address_in,
631         const int *interface_index_in);
632
633 /*i \ingroup NDDS_Transport_UDP_SocketFactoryClass
634  * \brief Default implementation of the socket creation factory method.
635  *
636  * @post socket is created and bound
637  *
638  * @see NDDS_Transport_UDP_SocketFactory_Receive_Socket_Create_Fcn
639  */
640 extern NDDS_Transport_DllExport RTI_INT32
641 NDDS_Transport_UDPv4_SocketFactory_create_receive_socket(
642         struct NDDS_Transport_UDPv4_SocketFactory *me,
643         NDDS_Transport_Port_t *port_inout,
644         const struct NDDS_Transport_UDPv4 *ip_in,
645         const NDDS_Transport_Address_t *multicast_address_in);
646
647 /*i \ingroup NDDS_Transport_UDP_SocketFactoryClass
648  * \brief Default implementation of the socket destroy factory method
649  * @see NDDS_Transport_UDP_SocketFactory_Socket_Destroy_Fcn
650  */
651 extern NDDS_Transport_DllExport RTI_INT32
652 NDDS_Transport_UDPv4_SocketFactory_destroy_socket(
653         struct NDDS_Transport_UDPv4_SocketFactory *me,
654         RTI_INT32 s_in);
655
656 /*i \ingroup NDDS_Transport_UDP_SocketFactoryClass
657  * \brief Used to bind a socket to the desired port
658  *
659  * @param s_in \b In. socket to be bound
660  *
661  * @param port_desired_in \b In. port number to bind to, use 0 to be
662  * assigned an ephemeral port.
663  *
664  * @return if successful, the port number bound to or 0 if
665  * unsuccessful.
666  */
667 extern NDDS_Transport_DllExport NDDS_Transport_Port_t
668 NDDS_Transport_UDPv4_Socket_bind(
669         RTI_INT32 s_in,
670         const NDDS_Transport_Port_t port_desired_in);
671
672 /*i \ingroup NDDS_Transport_UDP_SocketFactoryClass
673  * \brief Used to bind a socket to the desired port with specified IP
674  *
675  * @param s_in \b In. socket to be bound
676  *
677  * @param port_desired_in \b In. port number to bind to, use 0 to be
678  * assigned an ephemeral port.
679  *
680  * @param ip_in \b In. IP to bind
681  *
682  * @return if successful, the port number bound to or 0 if
683  * unsuccessful.
684  */
685 extern NDDS_Transport_DllExport NDDS_Transport_Port_t
686 NDDS_Transport_UDPv4_Socket_bind_with_ip(
687         RTI_INT32 s_in,
688         const NDDS_Transport_Port_t port_desired_in,
689         unsigned int ip_in);
690
691 /*i \ingroup NDDS_Transport_UDP_SocketFactoryClass
692  * \brief Default socket factory.
693  */
694 extern NDDS_Transport_DllVariable struct NDDS_Transport_UDPv4_SocketFactory
695         NDDS_Transport_UDPv4_g_socketFactory;
696
697 /*i \ingroup NDDS_Transport_UDPv4_Plugin
698  * \brief Realization of NDDS_Transport_Get_Class_Name_Fcn_cEA for IPv4
699  * transports.
700  *
```

```
701  * Returns the string "udpv4".
702  *
703  * @see NDDS_Transport_Get_Class_Name_Fcn_cEA for complete
704  * documentation.
705  */
706 #define NDDS_Transport_UDPv4_get_class_name_cEA \
707   NDDS_Transport_UDP_get_class_name_cEA
708
709 /*e \ingroup NDDS_Transport_UDPv4_Plugin
710  * \brief Realization of NDDS_Transport_String_To_Address_Fcn_cEA for IP
711  * transports.
712  *
713  * Converts a host name string to a IPv4 address.
714  *
715  * @param self  NOT USED. May be NULL.
716  *
717  * @param address_out \st_out The corresponding numerical value in IPv4
718  * format.
719  *
720  * @param address_in \st_in The name of the IPv4 address. It can be a
721  * dot notation name or a host name. If NULL, then the IP address of
722  * the localhost will be returned.
723  *
724  * @see NDDS_Transport_String_To_Address_Fcn_cEA for complete
725  * documentation.
726  */
727 #define NDDS_Transport_UDPv4_string_to_address_cEA \
728   NDDS_Transport_UDP_string_to_address_cEA
729
730 /* ================================================================= */
731 /*                     UDP/IP Transport Plugin                       */
732 /* ================================================================= */
733
734 /*e \ingroup NDDS_Transport_UDPv4_Plugin
735  * \brief Create an instance of a UDPv4 Transport Plugin.
736  *
737  * An application may create and register multiple instances of this
738  * Transport Plugin with \ndds. Multiple instances of the plugin may be useful
739  * for partitioning the network interfaces across multiple \ndds domains.
740  * However, note that the underlying transport, the operating system's IP layer,
741  * is still a "singleton."  For example, if a unicast transport has already
742  * bound to a port, and another unicast transport tries to bind to the same
743  * port, the second attempt will fail.
744  *
745  * The transport plugin honors the interface/multicast "white" and "black"
746  * lists specified in the NDDS_Transport_UDP_Property_t#parent:
747  * - NDDS_Transport_Property_t#allow_interfaces_list,
748  * - NDDS_Transport_Property_t#deny_interfaces_list,
749  * - NDDS_Transport_Property_t#allow_multicast_interfaces_list,
750  * - NDDS_Transport_Property_t#deny_multicast_interfaces_list
751  *
752  * The format of a string in these lists is assumed to be in standard IPv4
753  * dot notation, possibly containing wildcards. For example:
754  *       - 10.10.30.232
755  *       - 10.10.*.*
756  *       - 192.168.1.*
757  *       - etc.
758  * Strings not in the correct format will be ignored.
759  *
760  * @param property_in \st_in Desired behavior of this transport. May be
761  * NULL for default property.
762  *
763  * @return A UDPv4 Transport Plugin instance on success; or NULL on failure.
764  */
765 extern NDDS_Transport_DllExport NDDS_Transport_Plugin *NDDS_Transport_UDPv4_new(
766         const struct NDDS_Transport_UDPv4_Property_t *property_in);
767
768 /*e \ingroup NDDS_Transport_UDPv4_WAN_Plugin
769  * @brief Create an instance of a UDPv4_WAN Transport Plugin.
770  *
771  * An application may create and register multiple instances of this
772  * Transport Plugin with \ndds. Multiple instances of the plugin may be useful
773  * for partitioning the network interfaces across multiple \ndds domains.
774  * However, note that the underlying transport, the operating system's IP layer,
775  * is still a "singleton."  For example, if a unicast transport has already
776  * bound to a port, and another unicast transport tries to bind to the same
777  * port, the second attempt will fail.
778  *
779  * The transport plugin honors the interface "white" and "black"
780  * lists specified in the NDDS_Transport_UDP_Property_t#parent:
781  * - NDDS_Transport_Property_t#allow_interfaces_list,
```

```
782  * - NDDS_Transport_Property_t#deny_interfaces_list,
783  *
784  * The format of a string in these lists is assumed to be in standard IPv4
785  * dot notation, possibly containing wildcards. For example:
786  *       - 10.10.30.232
787  *       - 10.10.*.*
788  *       - 192.168.1.*
789  *       - etc.
790  * Strings not in the correct format will be ignored.
791  *
792  * @param property_in \st_in Desired behavior of this transport. May be
793  * NULL for default property.
794  *
795  * @return A UDPv4_WAN Transport Plugin instance on success; or NULL on failure.
796  */
797 extern NDDS_Transport_DllExport NDDS_Transport_Plugin *
798 NDDS_Transport_UDPv4_WAN_new(
799         const struct NDDS_Transport_UDPv4_WAN_Property_t *property_in);
800
801 /*i \ingroup NDDS_Transport_UDPv4_Plugin
802  * \brief Create an instance of a UDPv4 Transport Plugin.
803  *
804  * \copydoc NDDS_Transport_UDPv4_new
805  *
806  * @param socket_factory_in \st_in The factory that will create all the
807  * sockets to be used in the transport.  May be NULL if default factory
808  * is to be used.  Only used by RTI internal testing.
809  * End users should set this parameter to NULL.
810  *
811  * @param interface_filter_in \st_in A function that will be used to
812  * filter out network interfaces that this plugin can use. May be NULL if
813  * default interface listener is to be used. The default listener uses
814  * The NDDS_Transport_UDP_Property_t#parent's lists to select the interfaces.
815  * Only used by RTI internal testing. End users should set this
816  * parameter to NULL.
817  *
818  * @exception RTI_LOG_MALLOC_FAILURE_d RTI_LOG_CREATION_FAILURE_s
819  * RTI_LOG_PRECONDITION_FAILURE RTI_LOG_INIT_FAILURE_s
820  */
821 extern NDDS_Transport_DllExport NDDS_Transport_Plugin *
822 NDDS_Transport_UDPv4_newI(
823         const struct NDDS_Transport_UDPv4_Property_t *property_in,
824         struct NDDS_Transport_UDPv4_SocketFactory *socket_factory_in,
825         struct NDDS_Transport_UDPv4_InterfaceListener *interface_listener_in);
826
827 /*i \ingroup NDDS_Transport_UDPv4_WAN_Plugin
828  * @brief Create an instance of a UDPv4_WAN Transport Plugin.
829  *
830  * \copydoc NDDS_Transport_UDPv4_WAN_new
831  *
832  * @param socket_factory_in \st_in The factory that will create all the
833  * sockets to be used in the transport.  May be NULL if default factory
834  * is to be used.  Only used by RTI internal testing.
835  * End users should set this parameter to NULL.
836  *
837  * @param interface_filter_in \st_in A function that will be used to
838  * filter out network interfaces that this plugin can use. May be NULL if
839  * default interface listener is to be used. The default listener uses
840  * The NDDS_Transport_UDP_Property_t#parent's lists to select the interfaces.
841  * Only used by RTI internal testing. End users should set this
842  * parameter to NULL.
843  *
844  * @exception RTI_LOG_MALLOC_FAILURE_d RTI_LOG_CREATION_FAILURE_s
845  * RTI_LOG_PRECONDITION_FAILURE RTI_LOG_INIT_FAILURE_s
846  */
847 extern NDDS_Transport_DllExport NDDS_Transport_Plugin *
848 NDDS_Transport_UDPv4_WAN_newI(
849         const struct NDDS_Transport_UDPv4_WAN_Property_t *property_in,
850         struct NDDS_Transport_UDPv4_SocketFactory *socket_factory_in,
851         struct NDDS_Transport_UDPv4_InterfaceListener *interface_listener_in);
852
853
854 #define NDDS_Transport_UDPv4_get_receive_interfaces_cEA \
855   NDDS_Transport_UDP_get_receive_interfaces_cEA
856
857
858 #define NDDS_Transport_UDPv4_get_ignore_loopback_interface_status \
859   NDDS_Transport_UDP_get_ignore_loopback_interface_status
860
861
862 #define NDDS_Transport_UDPv4_is_local_address \
```

```
863   NDDS_Transport_UDP_is_local_address
864
865
866 #define NDDS_Transport_UDPv4_map_transport_priority \
867   NDDS_Transport_UDP_map_transport_priority
868
869 /*i \ingroup NDDS_Transport_UDPv4_Plugin
870   \brief Definition of IPv4 local host.
871 */
872 #define NDDS_TRANSPORT_UDPV4_ADDRESS_LOCALHOST \
873   { \
874     { \
875       0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 127, 0, 0, 1 \
876     } \
877   }
878
879
880 #define NDDS_TRANSPORT_UDPV4_UNBLOCK_MESSAGE_SIZE \
881   NDDS_TRANSPORT_UDP_UNBLOCK_MESSAGE_SIZE
882
883
884 #define NDDS_Transport_UDPv4_Statistics NDDS_Transport_UDP_Statistics
885
886
887 #define NDDS_Transport_UDPv4_get_statistics NDDS_Transport_UDP_get_statistics
888
889
890 #define NDDS_Transport_UDPv4_get_priority_range \
891   NDDS_Transport_UDP_get_priority_range
892
893 #define NDDS_Transport_UDPv4_Address_print NDDS_Transport_UDP_Address_print
894
895 extern NDDS_Transport_DllExport RTIBool
896 NDDS_Transport_UDPv4_WAN_Address_to_string(
897         const NDDS_Transport_Address_t *me,
898         char *buffer,
899         size_t buffer_size_in);
900
901 extern NDDS_Transport_DllExport void NDDS_Transport_UDPv4_WAN_Address_print(
902         const NDDS_Transport_Address_t *me,
903         const char *desc,
904         RTI_INT32 indent);
905
906
907 #ifdef __cplusplus
908 } /* extern "C" */
909 #endif
910
911 #include "transport/transport_udpv4_impl.h"
912
913 #endif /* transport_udpv4_h */
```

10.8 transport_udpv6.h

```
1 /*
2  * (c) Copyright, Real-Time Innovations, 2006-2024.
3  * All rights reserved.
4  *
5  * No duplications, whole or partial, manual or electronic, may be made
6  * without express written permission.  Any such copies, or
7  * revisions thereof, must display this notice unaltered.
8  * This code contains trade secrets of Real-Time Innovations, Inc.
9  */
10
11 #ifndef transport_udpv6_h
12 #define transport_udpv6_h
13
14 #include "transport/transport_udp.h"
15
16 #ifdef __cplusplus
17 extern "C" {
18 #endif
19
20 #if defined(RTI_VX653)
21 struct in6_addr;
22 #endif
23
```

```
24 #define NDDS_Transport_UDPv6 NDDS_Transport_UDP
25
26 #define NDDS_Transport_UDPv6_InterfaceListener \
27   NDDS_Transport_UDP_InterfaceListener
28
29 /* ================================================================= */
30 /*                Component: UDP/IPv6 Transport Plugin               */
31 /* ================================================================= */
32
33 /*e \dref_UDPv6TransportBuiltinGroupDocs
34  */
35
36 /*i \file
37  * \ingroup NDDS_Transport_UDPv6_Plugin
38  * \brief UDPv6 Transport Plugin interfaces and definitions.
39  */
40
41 /* ================================================================= */
42 /*                 UDPv6 Transport Properties                        */
43 /* ================================================================= */
44
45 #ifdef DOXYGEN_DOCUMENTATION_ONLY
46 /*e \dref_UDPv6Transport_Property_t
47  */
48 struct NDDS_Transport_UDPv6_Property_t {
49     /*e \dref_UDPv6Transport_Property_t_parent */
50     struct NDDS_Transport_Property_t parent;
51
52     /*e \dref_UDPv6Transport_Property_t_send_socket_buffer_size */
53     RTI_INT32 send_socket_buffer_size;
54
55     /*e \dref_UDPv6Transport_Property_t_recv_socket_buffer_size */
56     RTI_INT32 recv_socket_buffer_size;
57
58     /*e \dref_UDPv6Transport_Property_t_unicast_enabled */
59     RTI_INT32 unicast_enabled;
60
61     /*e \dref_UDPv6Transport_Property_t_multicast_enabled */
62     RTI_INT32 multicast_enabled;
63
64     /*e \dref_UDPv6Transport_Property_t_multicast_ttl */
65     RTI_INT32 multicast_ttl;
66
67     /*e \dref_UDPv6Transport_Property_t_multicast_loopback_disabled */
68     RTI_INT32 multicast_loopback_disabled;
69
70     /*e \dref_UDPv6Transport_Property_t_ignore_loopback_interface */
71     RTI_INT32 ignore_loopback_interface;
72
73     /*e \dref_UDPv6Transport_Property_t_ignore_nonrunning_interfaces */
74     RTI_INT32 ignore_nonrunning_interfaces;
75
76     /*e \dref_UDPv6Transport_Property_t_no_zero_copy */
77     RTI_INT32 no_zero_copy;
78
79     /*e \dref_UDPv6Transport_Property_t_send_blocking */
80     RTI_INT32 send_blocking;
81
82     /*e \dref_UDPv6Transport_Property_t_enable_v4mapped */
83     RTI_INT32 enable_v4mapped;
84
85     /*e \dref_UDPv6Transport_Property_t_transport_priority_mask */
86     RTI_UINT32 transport_priority_mask;
87
88     /*e \dref_UDPv6Transport_Property_t_transport_priority_mapping_low */
89     RTI_INT32 transport_priority_mapping_low;
90
91     /*e \dref_UDPv6Transport_Property_t_transport_priority_mapping_high */
92     RTI_INT32 transport_priority_mapping_high;
93
94     /*e \dref_UDPv6Transport_Property_t_send_ping */
95     RTI_INT32 send_ping;
96
97     /*e \dref_UDPv6Transport_Property_t_force_interface_poll_detection */
98     RTI_INT32 force_interface_poll_detection;
99
100     /*e \dref_UDPv6Transport_Property_t_interface_poll_period */
101     RTI_UINT32 interface_poll_period;
102
103     /*e \dref_UDPv6Transport_Property_t_reuse_multicast_receive_resource */
104     RTI_INT32 reuse_multicast_receive_resource;
```

```
105
106     /*e \dref_UDPv6Transport_Property_t_protocol_overhead_max */
107     RTI_INT32 protocol_overhead_max;
108
109     /*e \dref_UDPv6Transport_Property_t_disable_interface_tracking */
110     RTI_INT32 disable_interface_tracking;
111
112     /*e \dref_UDPv6Transport_Property_t_join_multicast_group_timeout */
113     RTI_UINT32 join_multicast_group_timeout;
114
115     /*e \dref_UDPv6Transport_Property_t_public_address */
116     char *public_address;
117
118     /*e \dref_UDPv6Transport_Property_t_port_offset */
119     RTI_INT32 port_offset;
120 };
121 #endif
122
123 #define NDDS_Transport_UDPv6_Property_t NDDS_Transport_UDP_Property_t
124
125 /*i \ingroup NDDS_Transport_UDPv6_Plugin
126  * \brief UDPv6 plugin class string name
127  */
128 #define NDDS_TRANSPORT_UDPV6_CLASS_NAME "udpv6"
129
130 /*i \ingroup NDDS_Transport_UDPv6_Plugin
131  * \brief UDPv6 builtin plugin alias
132  */
133 #define NDDS_TRANSPORT_BUILTIN_UDPV6_ALIAS "builtin.udpv6"
134
135 /*e \ingroup NDDS_Transport_UDPv6_Plugin
136  * \brief Default value of NDDS_Transport_Property_t#address_bit_count
137  */
138 #define NDDS_TRANSPORT_UDPV6_ADDRESS_BIT_COUNT (128)
139
140 /*e \ingroup NDDS_Transport_UDPv6_Plugin
141  * \brief Default value of NDDS_Transport_Property_t#properties_bitmap.
142  */
143 #define NDDS_TRANSPORT_UDPV6_PROPERTIES_BITMAP_DEFAULT \
144     NDDS_TRANSPORT_UDP_PROPERTIES_BITMAP_DEFAULT
145
146 /*e \ingroup NDDS_Transport_UDPv6_Plugin
147  * \brief Default value of
148  * NDDS_Transport_Property_t#gather_send_buffer_count_max.
149  *
150  * This is also the maximum value that can be used when instantiating the UDP
151  * transport.
152  *
153  * 16 is sufficient for NDDS, but more may improve discovery and reliable
154  * performance.
155  * Porting note: find out what the maximum gather buffer count is on your OS!
156  */
157 #define NDDS_TRANSPORT_UDPV6_GATHER_SEND_BUFFER_COUNT_MAX_DEFAULT \
158     NDDS_TRANSPORT_UDP_GATHER_SEND_BUFFER_COUNT_MAX_DEFAULT
159
160 /*e \dref_UDPv6Transport_SOCKET_BUFFER_SIZE_OS_MAX
161  */
162 #define NDDS_TRANSPORT_UDPV6_SOCKET_BUFFER_SIZE_OS_MAX \
163     NDDS_TRANSPORT_UDP_SOCKET_BUFFER_SIZE_OS_MAX
164
165 /*e \dref_UDPv6Transport_SOCKET_BUFFER_SIZE_OS_DEFAULT
166  */
167 #define NDDS_TRANSPORT_UDPV6_SOCKET_BUFFER_SIZE_OS_DEFAULT \
168     NDDS_TRANSPORT_UDP_SOCKET_BUFFER_SIZE_OS_DEFAULT
169
170 /*e \dref_UDPv6Transport_SEND_SOCKET_BUFFER_SIZE_DEFAULT
171  */
172 #define NDDS_TRANSPORT_UDPV6_SEND_SOCKET_BUFFER_SIZE_DEFAULT \
173     NDDS_TRANSPORT_UDP_SEND_SOCKET_BUFFER_SIZE_DEFAULT
174
175 /*e \dref_UDPv6Transport_RECV_SOCKET_BUFFER_SIZE_DEFAULT
176  */
177 #define NDDS_TRANSPORT_UDPV6_RECV_SOCKET_BUFFER_SIZE_DEFAULT \
178     NDDS_TRANSPORT_UDP_RECV_SOCKET_BUFFER_SIZE_DEFAULT
179
180 /*e \ingroup NDDS_Transport_UDPv6_Plugin
181   \brief Maximum value of NDDS_Transport_Property_t#message_size_max.
182  */
183 #define NDDS_TRANSPORT_UDPV6_PAYLOAD_SIZE_MAX (65487)
184
185 /*e \ingroup NDDS_Transport_UDPv6_Plugin
```

```
186   \brief Default value of NDDS_Transport_Property_t#message_size_max.
187  */
188 #if defined(RTI_INTY)
189   #define NDDS_TRANSPORT_UDPV6_MESSAGE_SIZE_MAX_DEFAULT (9196)
190 #else
191   #define NDDS_TRANSPORT_UDPV6_MESSAGE_SIZE_MAX_DEFAULT \
192     NDDS_TRANSPORT_UDPV6_PAYLOAD_SIZE_MAX
193 #endif
194
195 #define NDDS_TRANSPORT_UDPV6_DATAGRAM_LENGTH_MAX \
196   NDDS_TRANSPORT_UDP_DATAGRAM_LENGTH_MAX
197
198 /*e \ingroup NDDS_Transport_UDPv6_Plugin
199   \brief Default value of NDDS_Transport_UDPv6_Property_t#multicast_ttl.
200  */
201 #define NDDS_TRANSPORT_UDPV6_MULTICAST_TTL_DEFAULT \
202   NDDS_TRANSPORT_UDP_MULTICAST_TTL_DEFAULT
203
204 #define NDDS_TRANSPORT_UDPV6_USE_MULTICAST_DEFAULT \
205   NDDS_TRANSPORT_UDP_USE_MULTICAST_DEFAULT
206
207
208 /*e \dref_UDPv6Transport_IGNORE_LOOPBACK_DISABLED
209  */
210 #define NDDS_TRANSPORT_UDPV6_IGNORE_LOOPBACK_DISABLED \
211   NDDS_TRANSPORT_UDP_IGNORE_LOOPBACK_DISABLED
212
213 /*e \dref_UDPv6Transport_IGNORE_LOOPBACK_ENABLED
214  */
215 #define NDDS_TRANSPORT_UDPV6_IGNORE_LOOPBACK_ENABLED \
216   NDDS_TRANSPORT_UDP_IGNORE_LOOPBACK_ENABLED
217
218 /*e \dref_UDPv6Transport_IGNORE_LOOPBACK_AUTO
219  */
220 #define NDDS_TRANSPORT_UDPV6_IGNORE_LOOPBACK_AUTO \
221   NDDS_TRANSPORT_UDP_IGNORE_LOOPBACK_AUTO
222
223
224 /*e \dref_UDPv6Transport_BLOCKING_NEVER
225  */
226 #define NDDS_TRANSPORT_UDPV6_BLOCKING_NEVER NDDS_TRANSPORT_UDP_BLOCKING_NEVER
227
228 /*e \dref_UDPv6Transport_BLOCKING_ALWAYS
229  */
230 #define NDDS_TRANSPORT_UDPV6_BLOCKING_ALWAYS NDDS_TRANSPORT_UDP_BLOCKING_ALWAYS
231 #define NDDS_TRANSPORT_UDPV6_BLOCKING_UNICAST_ONLY \
232   NDDS_TRANSPORT_UDP_BLOCKING_UNICAST_ONLY
233
234 #define NDDS_TRANSPORT_UDPV6_BLOCKING_DEFAULT \
235   NDDS_TRANSPORT_UDP_BLOCKING_DEFAULT
236
237
238 #define NDDS_TRANSPORT_UDPV6_PROPERTY_DEFAULT \
239     { \
240         { \
241             NDDS_TRANSPORT_CLASSID_UDPv6, \
242             NDDS_TRANSPORT_UDPV6_ADDRESS_BIT_COUNT, \
243             NDDS_TRANSPORT_UDPV6_PROPERTIES_BITMAP_DEFAULT, \
244             NDDS_TRANSPORT_UDPV6_GATHER_SEND_BUFFER_COUNT_MAX_DEFAULT, \
245             NDDS_TRANSPORT_UDPV6_MESSAGE_SIZE_MAX_DEFAULT, \
246             NULL, \
247             0, /* allow_interfaces_list */ \
248             NULL, \
249             0, /* deny_interfaces_list */ \
250             NULL, \
251             0, /* allow_multicast_interfaces_list */ \
252             NULL, \
253             0,                          /* deny_multicast_interfaces_list */ \
254             NDDS_TRANSPORT_UUID_UNKNOWN, /* transport_uuid */ \
255             NULL,                       /* domain_participant_ptr */ \
256             NULL,                       /* domain_participant_forwarder */ \
257             NULL,                       /* thread_name_prefix */ \
258             NDDS_TRANSPORT_LENGTH_UNLIMITED, /* max_interface_count */ \
259         }, \
260                 NDDS_TRANSPORT_UDPV6_SEND_SOCKET_BUFFER_SIZE_DEFAULT, \
261                 NDDS_TRANSPORT_UDPV6_RECV_SOCKET_BUFFER_SIZE_DEFAULT, \
262                 1,                                          /* use \
263                                                                unicast \
264                                                             */ \
265                 NDDS_TRANSPORT_UDPV6_USE_MULTICAST_DEFAULT, /* use multicast \
266                                                             */ \
```

```
267                 NDDS_TRANSPORT_UDPV6_MULTICAST_TTL_DEFAULT, 0, /* multicast \
268                                                                   loopback \
269                                                                   enabled */ \
270                 NDDS_TRANSPORT_UDPV6_IGNORE_LOOPBACK_AUTO, /* (auto-)ignore \
271                                                               loopback */ \
272                 1, /* ignore_nonup_interfaces */ \
273                 1, /* ignore_nonrunning_interfaces */ \
274                 0, /* no_zero_copy */ \
275                 NDDS_TRANSPORT_UDPV6_BLOCKING_DEFAULT, 0, /* enable_v4mapped \
276                                                              */ \
277                 0, /* use_checksum (no effect, udpv4 only) */ \
278                 0, 0, 0xff /* no mapping to IP_TOS by default */, \
279                 1,    /* send_ping \
280                        */ \
281                 0,    /* don't force polling interface tracker by default */ \
282                 500,  /* 500 millisecs is the default polling period*/ \
283                 1,    /* reuse multicast receive resource */ \
284                 48,   /* protocol_overhead_max (UDP(8)+IP(40)) */ \
285                 0,    /* disable interface tracking */ \
286                 5000, /* join_multicast_group_timeout */ \
287                 NULL, /* public_address */ \
288                 NULL, 0,   /* comm_ports_list */ \
289                 NULL,      /* wan_listener */ \
290                 0,         /* port_offset */ \
291                 0,         /* binding_ping_period */ \
292                 0,         /* binding_ping_mapping_expiration */ \
293                 NULL,      /* plugin_enabled_function_ptr */ \
294                 RTI_FALSE, /* gather_detailed_statistics */ \
295                 RTI_FALSE, /* drop_samples_by_guid_prefix */ \
296                 RTI_OSAPI_RTPS_GUID_PREFIX_UNKNOWN /* guid_prefix_to_drop */ \
297     }
298
299 /* ================================================================= */
300 /*                  InterfaceListener Class                          */
301 /* ================================================================= */
302
303 #define NDDS_Transport_UDPv6_InterfaceListener \
304   NDDS_Transport_UDP_InterfaceListener
305
306
307 #define NDDS_Transport_UDPv6_Interface_Filter_Fcn \
308   NDDS_Transport_UDP_Interface_Filter_Fcn
309
310 /* ================================================================= */
311 /*                    UDP/IPv6 InterfaceListener                     */
312 /* ================================================================= */
313
314
315 /*i \ingroup NDDS_Transport_UDPv6_InterfaceListenerClass
316  * \brief Default implementation of the Interface listener.
317  *
318  * The default interface listener applies the interface selection preferences
319  * specified in the NDDS_Transport_UDPv6_Property_t#parent:
320  * - NDDS_Transport_Property_t#allow_interfaces_list,
321  * - NDDS_Transport_Property_t#deny_interfaces_list,
322  * - NDDS_Transport_Property_t#allow_multicast_interfaces_list,
323  * - NDDS_Transport_Property_t#deny_multicast_interfaces_list
324  *
325  * The filter assumes that the strings passed in the address filter
326  * are in UDPv6 dot notation, possibly containing wildcards. For example:
327  *        ffce:2222:*:0000
328  *        0000:*:0001
329  *        0*1  etc.
330  *
331  *        0::192.168.1.*   (TODO: this form is not yet supported)
332  * etc.
333  *
334  * @see NDDS_Transport_UDPv6_Interface_Filter_Fcn
335  */
336 extern NDDS_Transport_DllExport RTI_INT32
337 NDDS_Transport_UDPv6_InterfaceListener_onInterface(
338         struct NDDS_Transport_UDPv6_InterfaceListener *self,
339         struct NDDS_Transport_UDPv6 *plugin,
340         RTI_INT32 *multicast_enabled_out,
341         NDDS_Transport_Interface_t *interface_in,
342         const char *interface_name_in);
343
344 /*i \ingroup NDDS_Transport_UDPv6_InterfaceListenerClass
345  * \brief Default interface listener.
346  */
347 extern NDDS_Transport_DllVariable struct NDDS_Transport_UDP_InterfaceListener
```

```
348         NDDS_Transport_UDPv6_g_interfaceListener;
349
350 /* ================================================================= */
351 /*                   Socket Factory Class                            */
352 /* ================================================================= */
353 /*i
354  * \defgroup NDDS_Transport_UDPv6_SocketFactoryClass
355  * NDDS_Transport_UDPv6_SocketFactory \ingroup NDDS_Transport_UDPv6_Plugin
356  * \brief Interface for IPv6 socket creation and destruction.
357  *
358  * @mtsafety See NDDS_Transport_UDPv6_public UML diagram
359  *
360  * @see NDDS_Transport_UDPv6_new
361  */
362
363 #define NDDS_Transport_UDPv6_SocketFactory_Send_Socket_Create_Fcn \
364   NDDS_Transport_UDP_SocketFactory_Send_Socket_Create_Fcn
365
366 #define NDDS_Transport_UDPv6_SocketFactory_Receive_Socket_Create_Fcn \
367   NDDS_Transport_UDP_SocketFactory_Receive_Socket_Create_Fcn
368
369 #define NDDS_Transport_UDPv6_SocketFactory_Socket_Destroy_Fcn \
370   NDDS_Transport_UDP_SocketFactory_Socket_Destroy_Fcn
371
372 #define NDDS_Transport_UDPv6_SocketFactory NDDS_Transport_UDP_SocketFactory
373
374 #define NDDS_Transport_UDPv6_SocketFactory_is_valid \
375   NDDS_Transport_UDP_SocketFactory_is_valid
376
377 /* ================================================================= */
378 /*                 IP Socket Factory Class                           */
379 /* ================================================================= */
380
381 /*i \ingroup NDDS_Transport_UDPv6_SocketFactoryClass
382  * \brief Default implementation of the socket creation factory method
383  * for UDPv6 transport.
384  *
385  * @post Sending socket is not bound.
386  *
387  * @see NDDS_Transport_UDPv6_SocketFactory_Send_Socket_Create_Fcn
388  */
389 extern NDDS_Transport_DllExport RTI_INT32
390 NDDS_Transport_UDPv6_SocketFactory_create_send_socket(
391         struct NDDS_Transport_UDPv6_SocketFactory *me,
392         NDDS_Transport_Port_t *port_out,
393         const struct NDDS_Transport_UDPv6 *ip_in,
394         const NDDS_Transport_Address_t *interface_address_in,
395         const int *interface_index_in);
396
397 /*i \ingroup NDDS_Transport_UDPv6_SocketFactoryClass
398  * \brief Default implementation of the socket creation factory method.
399  *
400  * @post socket is created and bound
401  *
402  * @see NDDS_Transport_UDPv6_SocketFactory_Receive_Socket_Create_Fcn
403  */
404 extern NDDS_Transport_DllExport RTI_INT32
405 NDDS_Transport_UDPv6_SocketFactory_create_receive_socket(
406         struct NDDS_Transport_UDPv6_SocketFactory *me,
407         NDDS_Transport_Port_t *port_inout,
408         const struct NDDS_Transport_UDPv6 *ip_in,
409         const NDDS_Transport_Address_t *multicast_address_in);
410
411 /*i \ingroup NDDS_Transport_UDPv6_SocketFactoryClass
412  * \brief Default implementation of the socket destroy factory method
413  * @see NDDS_Transport_UDPv6_SocketFactory_Socket_Destroy_Fcn
414  */
415 extern NDDS_Transport_DllExport RTI_INT32
416 NDDS_Transport_UDPv6_SocketFactory_destroy_socket(
417         struct NDDS_Transport_UDPv6_SocketFactory *me,
418         RTI_INT32 s_in);
419
420 /*i \ingroup NDDS_Transport_UDPv6_SocketFactoryClass
421  * \brief Used to bind a socket to the desired port
422  *
423  * @param s_in \b In. socket to be bound
424  *
425  * @param port_desired_in \b In. port number to bind to, use 0 to be
426  * assigned an ephemeral port.
427  *
428  * @return if successful, the port number bound to or 0 if
```

```
429  * unsuccessful.
430  */
431 extern NDDS_Transport_DllExport NDDS_Transport_Port_t
432 NDDS_Transport_UDPv6_Socket_bind(
433         RTI_INT32 s_in,
434         const NDDS_Transport_Port_t port_desired_in);
435
436 /*i \ingroup NDDS_Transport_UDPv6_SocketFactoryClass
437  * \brief Used to bind a socket to the desired port with specified IP
438  *
439  * @param s_in \b In. socket to be bound
440  *
441  * @param port_desired_in \b In. port number to bind to, use 0 to be
442  * assigned an ephemeral port.
443  *
444  * @param ip_in \b In. IP to bind
445  *
446  * @return if successful, the port number bound to or 0 if
447  * unsuccessful.
448  */
449 extern NDDS_Transport_DllExport NDDS_Transport_Port_t
450 NDDS_Transport_UDPv6_Socket_bind_with_ip(
451         RTI_INT32 s_in,
452         const NDDS_Transport_Port_t port_desired_in,
453         const struct in6_addr *ip_in);
454
455 /*i \ingroup NDDS_Transport_UDPv6_SocketFactoryClass
456  * \brief Default socket factory.
457  */
458 extern NDDS_Transport_DllVariable struct NDDS_Transport_UDPv6_SocketFactory
459         NDDS_Transport_UDPv6_g_socketFactory;
460
461 #define NDDS_Transport_UDPv6_get_class_name_cEA \
462   NDDS_Transport_UDP_get_class_name_cEA
463
464 /*e \ingroup NDDS_Transport_UDPv6_Plugin
465  * \brief Realization of NDDS_Transport_String_To_Address_Fcn_cEA for IP
466  * transports.
467  *
468  * Converts a host name string to a IPv6 address.
469  *
470  * @param self  NOT USED. May be NULL.
471  *
472  * @param address_out \st_out The corresponding numerical value in IPv6
473  * format.
474  *
475  * @param address_in \st_in The name of the IPv6 address. It can be a
476  * dot notation name or a host name. If NULL, then the IP address of
477  * the localhost will be returned.
478  *
479  * @see NDDS_Transport_String_To_Address_Fcn_cEA for complete
480  * documentation.
481  */
482 #define NDDS_Transport_UDPv6_string_to_address_cEA \
483   NDDS_Transport_UDP_string_to_address_cEA
484
485 /* ================================================================= */
486 /*                     UDP/IPv6 Transport Plugin                     */
487 /* ================================================================= */
488
489 /*e \ingroup NDDS_Transport_UDPv6_Plugin
490  * \brief Create an instance of a UDPv6 Transport Plugin.
491  *
492  * An application may create and register multiple instances of this
493  * Transport Plugin with \ndds. Multiple instances of the plugin may be useful
494  * for partitioning the network interfaces across multiple \ndds domains.
495  * However, note that the underlying transport, the operating system's IP layer,
496  * is still a "singleton."  For example, if a unicast transport has already
497  * bound to a port, and another unicast transport tries to bind to the same
498  * port, the second attempt will fail.
499  *
500  * The transport plugin honors the interface/multicast "white" and "black"
501  * lists specified in the NDDS_Transport_UDPv6_Property_t#parent:
502  * - NDDS_Transport_Property_t#allow_interfaces_list,
503  * - NDDS_Transport_Property_t#deny_interfaces_list,
504  * - NDDS_Transport_Property_t#allow_multicast_interfaces_list,
505  * - NDDS_Transport_Property_t#deny_multicast_interfaces_list
506  *
507  * The format of a string in these lists is assumed to be in standard IPv6
508  * dot notation, possibly containing wildcards. For example:
509  *       - 10.10.30.232
```

```
510  *       - 10.10.*.*
511  *       - 192.168.1.*
512  *       - etc.
513  * Strings not in the correct format will be ignored.
514  *
515  * @param property_in \st_in Desired behavior of this transport. May be
516  * NULL for default property.
517  *
518  * @return A UDPv6 Transport Plugin instance on success; or NULL on failure.
519  */
520 extern NDDS_Transport_DllExport NDDS_Transport_Plugin *NDDS_Transport_UDPv6_new(
521         const struct NDDS_Transport_UDPv6_Property_t *property_in);
522
523 /*i \ingroup NDDS_Transport_UDPv6_Plugin
524  * \brief Create an instance of a UDPv6 Transport Plugin.
525  *
526  * \copydoc NDDS_Transport_UDPv6_new
527  *
528  * @param socket_factory_in \st_in The factory that will create all the
529  * sockets to be used in the transport.  May be NULL if default factory
530  * is to be used.  Only used by RTI internal testing.
531  * End users should set this parameter to NULL.
532  *
533  * @param interface_filter_in \st_in A function that will be used to
534  * filter out network interfaces that this plugin can use. May be NULL if
535  * default interface listener is to be used. The default listener uses
536  * The NDDS_Transport_UDPv6_Property_t#parent's lists to select the interfaces.
537  * Only used by RTI internal testing. End users should set this
538  * parameter to NULL.
539  *
540  * @exception RTI_LOG_MALLOC_FAILURE_d RTI_LOG_CREATION_FAILURE_s
541  * RTI_LOG_PRECONDITION_FAILURE RTI_LOG_INIT_FAILURE_s
542  */
543 extern NDDS_Transport_DllExport NDDS_Transport_Plugin *
544 NDDS_Transport_UDPv6_newI(
545         const struct NDDS_Transport_UDPv6_Property_t *property_in,
546         struct NDDS_Transport_UDPv6_SocketFactory *socket_factory_in,
547         struct NDDS_Transport_UDPv6_InterfaceListener *interface_listener_in);
548
549 #define NDDS_Transport_UDPv6_get_receive_interfaces_cEA \
550   NDDS_Transport_UDP_get_receive_interfaces_cEA
551
552 #define NDDS_Transport_UDPv6_get_ignore_loopback_interface_status \
553   NDDS_Transport_UDP_get_ignore_loopback_interface_status
554
555 #define NDDS_Transport_UDPv6_is_local_address \
556   NDDS_Transport_UDP_is_local_address
557
558 #define NDDS_Transport_UDPv6_map_transport_priority \
559   NDDS_Transport_UDP_map_transport_priority
560
561 /*i \ingroup NDDS_Transport_UDPv6_Plugin
562  * \brief Definition of local host.
563  */
564 #define NDDS_TRANSPORT_UDPV6_ADDRESS_LOCALHOST \
565   { \
566     { \
567       0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1 \
568     } \
569   }
570
571 #define NDDS_TRANSPORT_UDPV6_UNBLOCK_MESSAGE_SIZE \
572   NDDS_TRANSPORT_UDP_UNBLOCK_MESSAGE_SIZE
573
574 #define NDDS_Transport_UDPv6_Statistics NDDS_Transport_UDP_Statistics
575
576 #define NDDS_Transport_UDPv6_get_statistics NDDS_Transport_UDP_get_statistics
577
578 #define NDDS_Transport_UDPv6_get_priority_range \
579   NDDS_Transport_UDP_get_priority_range
580
581 #define NDDS_Transport_UDPv6_Address_print NDDS_Transport_UDP_Address_print
582
583
584 #ifdef __cplusplus
585 } /* extern "C" */
586 #endif
587
588 #include "transport/transport_udpv6_impl.h"
589
590 #endif /* transport_udpv6_h */
```

10.9 AggregationBuilders.hpp

```
1 /*
2 (c) Copyright, Real-Time Innovations, 2018.
3 All rights reserved.
4
5 No duplications, whole or partial, manual or electronic, may be made
6 without express written permission.  Any such copies, or
7 revisions thereof, must display this notice unaltered.
8 This code contains trade secrets of Real-Time Innovations, Inc.
9 */
10
11 #ifndef RTI_DDS_FLAT_AGGREGATIONBUILDERS_HPP_
12 #define RTI_DDS_FLAT_AGGREGATIONBUILDERS_HPP_
13
14 #include "rti/flat/BuilderHelper.hpp"
15 #include "rti/flat/Builder.hpp"
16 #include "xcdr/xcdr_interpreter.h"
17
18 namespace rti { namespace flat {
19
20 /*
21  * Parameters describing a member to be added or built.
22  *
23  * The information encoded by this structure is used to set the appropriate
24  * bits in the header that precedes each member of a mutable type.
25  */
26 struct MemberParams {
27     /*
28      * Indicates whether the member is part of the key.
29      */
30     bool key;
31     /*
32      * Indicates whether the member is the discriminator of a union.
33      */
34     bool discriminator;
35     /*
36      * Indicates whether the member has been marked as "must understand".
37      */
38     bool must_understand;
39
40     MemberParams() : key(false), discriminator(false), must_understand(false)
41     { }
42
43     /*
44      * Check whether the current XCDR2 compliance mask allows the
45      * specified features.
46      *
47      * This function will cache the value of the compliance mask to avoid
48      * repeated nested function calls at runtime.
49      *
50      * @param check_mask The feature bits to check in the compliance mask.
51      * @param reload_mask If true, force reloading of the compliance mask
52      * from the global configuration.
53      * @return true if all the bits in check_mask are set in the compliance
54      * mask.
55      */
56     static bool xcdr2_compliant(
57             const RTIXCdrXTypesComplianceMask check_mask,
58             const bool reload_mask)
59     {
60         static bool g_compliance_mask_set = false;
61         static RTIXCdrXTypesComplianceMask g_compliance_mask = 0;
62
63         // Initialize the compliance mask on first use
64         // There could be a data race here but the worst that can happen is
65         // that the compliance mask is read more than once.
66         // The goal is to avoid nested function calls on each and every call.
67         if (reload_mask || !g_compliance_mask_set) {
68             g_compliance_mask =
69                     RTIXCdrInterpreter_getGlobalXtypeComplianceMask();
70             g_compliance_mask_set = true;
71         }
72
73         return (g_compliance_mask & check_mask);
74     }
75
76     /*
77      * Check if the must_understand bit should be set in the header for this
78      * member.
```

```
79      *
80      * The bit will always be set if the must_understand attribute is true.
81      * Additionally, if the current XCDR2 compliance mask allows it, the bit
82      * will also be set for key members and discriminators.
83      *
84      * @return true if the must_understand bit should be set.
85      */
86     bool has_must_understand_bit() const
87     {
88         return must_understand
89                 || (xcdr2_compliant(
90                         RTI_XCDR_XTYPES_MUST_UNDERSTAND_IN_KEY_AND_DISCRIMINATOR_BIT,
91                         false)
92                     && (key || discriminator));
93     }
94 };
95
96 /*
97  * Provides the common functionality for builders of aggregated types
98  *
99  * Extends and specializes AbstractBuilder to add or override the functionality
100  * required to build an aggregated type:
101  *
102  * - Add or build members, prepending the member header
103  * - Add dheader at the beginning
104  * - Add parameter-list-end header at the end
105  */
114 class AggregationBuilder : public AbstractBuilder {
115 protected:
116     AggregationBuilder() :
117             dheader_position_(NULL),
118             emheader_position_(NULL)
119     {
120     }
121
122     // Create a new top-level builder
123     AggregationBuilder(
124             unsigned char *initial_buffer,
125             offset_t size,
126             bool initialize_members) :
127         AbstractBuilder(initial_buffer, size, initialize_members),
128         dheader_position_(NULL),
129         emheader_position_(NULL)
130     {
131         dheader_position_ = stream().serialize_dheader();
132         if (dheader_position_ == NULL) {
133             RTI_FLAT_BUILDER_OUT_OF_RESOURCES_ERROR(invalidate());
134         }
135     }
136
137 public:
138     // For internal use and testing
139     //
140     // Modifies the encapsulation set in the constructor with a custom value.
141     // This function can only be called after the construction of a non-nested
142     // Builder, before any member is built or added.
143     //
144     void set_encapsulation_impl(RTIXCdrEncapsulationId encapsulation_id)
145     {
146         // Only a top-level Builder can change the encapsulation
147         RTI_FLAT_CHECK_PRECONDITION(!is_nested(), return);
148
149         char *current_position = (char *) owned_stream_.current_position();
150         // This operation is only valid right after construction, when no
151         // member has been added yet
152         RTI_FLAT_CHECK_PRECONDITION(
153                 current_position == dheader_position_ + RTI_XCDR_DHEADER_SIZE,
154                 return);
155
156         owned_stream_.skip_back(
157                 RTI_XCDR_DHEADER_SIZE + RTI_XCDR_ENCAPSULATION_HEADER_SIZE);
158
159         if (!RTIXCdrFlatSample_initializeEncapsulationAndStream(
160                 (char *) owned_stream_.buffer(),
161                 &owned_stream_.c_stream(),
162                 encapsulation_id,
163                 owned_stream_.total_size())) {
164             RTI_FLAT_BUILDER_OUT_OF_RESOURCES_ERROR(return);
165         }
166
167         stream().serialize_dheader();
```

```
168     }
169
170 protected:
171
172 #if defined(RTI_FLAT_DATA_CXX11_RVALUE_REFERENCES)
173     AggregationBuilder(AggregationBuilder&& other) = default;
174
175     AggregationBuilder& operator=(AggregationBuilder&& other)
176     {
177         if (this == &other) {
178             return *this;
179         }
180
181         finish_untyped_impl();
182
183         AbstractBuilder::operator=(static_cast<AbstractBuilder&&>(other));
184
185         dheader_position_ = other.dheader_position_;
186         emheader_position_ = other.emheader_position_;
187
188         return *this;
189     }
190 #else
191 public:
192     // Enable the safe-move-constructor idiom without C++11 move constructors
193     struct UntypedAggregationBuilderMoveProxy : AbstractBuilderMoveProxy {
194         char * dheader_position_;
195         char * emheader_position_;
196     };
197
198     operator UntypedAggregationBuilderMoveProxy () throw() // move-constructor idiom
199     {
200         UntypedAggregationBuilderMoveProxy other;
201         move_to(other);
202         return other;
203     }
204
205 protected:
206     void move_from(UntypedAggregationBuilderMoveProxy& other)
207     {
208         AbstractBuilder::move_from(other);
209         dheader_position_ = other.dheader_position_;
210         emheader_position_ = other.emheader_position_;
211     }
212
213     void move_to(UntypedAggregationBuilderMoveProxy& other)
214     {
215         AbstractBuilder::move_to(other);
216         other.dheader_position_ = dheader_position_;
217         other.emheader_position_ = emheader_position_;
218     }
219 #endif
220
221 protected:
222     virtual ~AggregationBuilder()
223     {
224         // AggregationBuilder::finish_untyped_impl() is specialized
225         // with respect to AbstractBuilder::finish_untyped_impl()
226         finish_untyped_impl();
227     }
228
229     template <typename T>
230     bool add_primitive_member(
231             member_id_t id,
232             T value,
233             const MemberParams& params)
234     {
235         RTI_FLAT_BUILDER_CHECK_VALID(return false);
236
237         rti::xcdr::Stream::Memento stream_memento(stream());
238
239         if (!begin_emheader(id, detail::primitive_lc_code<T>::value, params)) {
240             RTI_FLAT_BUILDER_OUT_OF_RESOURCES_ERROR(return false);
241         }
242         // no need to call finish_emheader()
243
244         if (!stream().serialize_no_align(value)) {
245             RTI_FLAT_BUILDER_OUT_OF_RESOURCES_ERROR(return false);
246         }
247
248         stream_memento.discard(); // success: do not restore the current position
```

```
249         return true;
250     }
251
252     template <typename T>
253     bool add_primitive_member(member_id_t id, T value)
254     {
255         static const MemberParams default_params;
256         return add_primitive_member(id, value, default_params);
257     }
258
259     template <typename OffsetType>
260     OffsetType add_member(member_id_t id, const MemberParams& params)
261     {
262         RTI_FLAT_BUILDER_CHECK_VALID(return OffsetType());
263
264         // save state in case of error
265         rti::xcdr::Stream::Memento stream_memento(stream());
266         return add_member<OffsetType>(id, stream_memento, params);
267     }
268
269     template <typename OffsetType>
270     OffsetType add_member(member_id_t id)
271     {
272         RTI_FLAT_BUILDER_CHECK_VALID(return OffsetType());
273
274         // save state in case of error
275         rti::xcdr::Stream::Memento stream_memento(stream());
276         static const MemberParams default_params;
277         return add_member<OffsetType>(id, stream_memento, default_params);
278     }
279
280     template <typename OffsetType>
281     OffsetType add_member(
282             member_id_t id,
283             rti::xcdr::Stream::Memento& stream_memento)
284     {
285         static const MemberParams default_params;
286         return add_member<OffsetType>(id, stream_memento, default_params);
287     }
288
289     template <typename OffsetType>
290     OffsetType add_member(
291             member_id_t id,
292             rti::xcdr::Stream::Memento& stream_memento,
293             const MemberParams& params)
294     {
295         // fixed-size non-primitive members may have a dheader or not; when they
296         // do the LC code is 5, otherwise it is 4.
297         if (!begin_emheader(
298                     id,
299                     detail::is_fixed_type_w_dheader<OffsetType>::value() ? 5
300                                                                          : 4,
301                     params)) {
302             return OffsetType();
303         }
304
305         OffsetType result = AbstractBuilder::add_element<OffsetType>();
306         finish_emheader();
307
308         stream_memento.discard(); // success - do not restore the previous state
309         return result;
310     }
311
312     template <typename NestedBuilder>
313     NestedBuilder build_member(member_id_t id, const MemberParams& params)
314     {
315         RTI_FLAT_BUILDER_CHECK_VALID(return NestedBuilder());
316
317         // save state in case of error
318         rti::xcdr::Stream::Memento stream_memento(stream());
319         return build_member<NestedBuilder>(id, stream_memento, params);
320     }
321
322     template <typename NestedBuilder>
323     NestedBuilder build_member(member_id_t id)
324     {
325         static const MemberParams default_params;
326         return build_member<NestedBuilder>(id, default_params);
327     }
328
329     template <typename NestedBuilder>
```

```
330     NestedBuilder build_member(
331             member_id_t id,
332             rti::xcdr::Stream::Memento& stream_memento)
333     {
334         static const MemberParams default_params;
335         return build_member<NestedBuilder>(id, stream_memento, default_params);
336     }
337
338     template <typename NestedBuilder>
339     NestedBuilder build_member(
340             member_id_t id,
341             rti::xcdr::Stream::Memento& stream_memento,
342             const MemberParams& params)
343     {
344         if (!begin_emheader(
345                     id,
346                     static_cast<RTIXCdrUnsignedLong>(
347                             detail::lc_code<NestedBuilder>::value()),
348                     params)) {
349             return NestedBuilder();
350         }
351
352         return AbstractBuilder::build_element_no_align<NestedBuilder>(
353                 stream_memento);
354     }
355
356     unsigned char* finish_sample_impl()
357     {
358         RTI_FLAT_BUILDER_CHECK_VALID(return NULL);
359         RTI_FLAT_BUILDER_CHECK_NOT_BOUND(return NULL);
360         RTI_FLAT_BUILDER_CHECK_CAN_FINISH_SAMPLE(return NULL);
361
362         if (!stream().finish_dheader(dheader_position_)) {
363             RTI_FLAT_BUILDER_OUT_OF_RESOURCES_ERROR(return NULL);
364         }
365
366         unsigned char *return_sample = stream().buffer();
367         invalidate();
368
369         return return_sample;
370     }
371
372     // Same as finish_impl() but it doesn't throw exceptions or return an Offset
373     void finish_untyped_impl()
374     {
375         if (is_valid() && is_nested()) {
376             stream().finish_dheader(dheader_position_);
377             AbstractBuilder::finish_untyped_impl();
378         }
379     }
380
381     template <typename OffsetType>
382     OffsetType finish_impl()
383     {
384         RTI_FLAT_BUILDER_CHECK_VALID(return OffsetType());
385         RTI_FLAT_BUILDER_CHECK_CAN_FINISH(return OffsetType());
386
387         stream().finish_dheader(dheader_position_);
388         return AbstractBuilder::finish_impl<OffsetType>();
389     }
390
391 public:
392     // For internal use; need to use rti::flat::discard_sample(writer, builder)
393     unsigned char * discard_sample_impl()
394     {
395         RTI_FLAT_BUILDER_CHECK_VALID(return NULL);
396         RTI_FLAT_BUILDER_CHECK_NOT_BOUND(return NULL);
397         RTI_FLAT_BUILDER_CHECK_CAN_FINISH_SAMPLE(return NULL);
398
399         unsigned char *buffer = this->buffer();
400         invalidate();
401         return buffer;
402     }
403
404 private:
405     void finish_member() // override noexcept
406     {
407         if (!is_valid()) {
408             return;
409         }
410
```

```
411         finish_emheader();
412         AbstractBuilder::finish_member();
413     }
414
415 private:
416     bool begin_emheader(
417             member_id_t id,
418             RTIXCdrUnsignedLong lc,
419             const MemberParams& params)
420     {
421         RTI_FLAT_BUILDER_CHECK_NOT_BOUND(return false);
422
423         if (!stream().serialize_emheader(
424                     emheader_position_,
425                     id,
426                     lc,
427                     params.has_must_understand_bit())) {
428             RTI_FLAT_BUILDER_OUT_OF_RESOURCES_ERROR(return false);
429         }
430
431         return true;
432     }
433
434     void finish_emheader() // noexcept
435     {
436         if (emheader_position_ != NULL) {
437             stream().finish_emheader(emheader_position_);
438             emheader_position_ = NULL;
439         }
440     }
441
442 protected:
443     // Create a nested builder
444     AggregationBuilder(nested_tag_t, AbstractBuilder& parent, unsigned int alignment)
445         : AbstractBuilder(nested_tag_t(), parent, alignment),
446           dheader_position_(NULL),
447           emheader_position_(NULL)
448     {
449         dheader_position_ = stream().serialize_dheader();
450         if (dheader_position_ == NULL) {
451             RTI_FLAT_BUILDER_OUT_OF_RESOURCES_ERROR(invalidate());
452         }
453     }
454
455 private:
456     char * dheader_position_;
457     char * emheader_position_;
458 };
459
460 // Specializes an AggregationBuilder to enforce that only one member can
461 // be set at a time
471 template <typename Discriminator>
472 class UnionBuilder : public AggregationBuilder {
473 protected:
474     UnionBuilder()
475     {
476     }
477
478     // Create a new top-level builder
479     UnionBuilder(
480             unsigned char *initial_buffer,
481             offset_t size,
482             bool initialize_members) :
483         AggregationBuilder(initial_buffer, size, initialize_members)
484     {
485     }
486
487     template <typename T>
488     bool add_primitive_member(member_id_t id, Discriminator disc, T value)
489     {
490         RTI_FLAT_BUILDER_CHECK_VALID(return false);
491
492         // Always start at the beginning and override any member or discriminator
493         // that may have been set
494         move_to_discriminator();
495         rti::xcdr::Stream::Memento stream_memento(stream());
496
497         if (!add_discriminator(disc)) {
498             return false;
499         }
500
```

```
501         if (!AggregationBuilder::add_primitive_member(id, value)) {
502             return false;
503         }
504
505         stream_memento.discard(); // success: do not restore the current position
506         return true;
507     }
508
509     template <typename OffsetType>
510     OffsetType add_member(member_id_t id, Discriminator disc)
511     {
512         RTI_FLAT_BUILDER_CHECK_VALID(return OffsetType());
513
514         move_to_discriminator();
515         rti::xcdr::Stream::Memento stream_memento(stream());
516         if (!add_discriminator(disc)) {
517             RTI_FLAT_BUILDER_OUT_OF_RESOURCES_ERROR(return OffsetType());
518         }
519
520         return AggregationBuilder::add_member<OffsetType>(id, stream_memento);
521     }
522
523     template <typename NestedBuilder>
524     NestedBuilder build_member(member_id_t id, Discriminator disc)
525     {
526         RTI_FLAT_BUILDER_CHECK_VALID(return NestedBuilder());
527
528         move_to_discriminator();
529         rti::xcdr::Stream::Memento stream_memento(stream());
530         add_discriminator(disc);
531
532         return AggregationBuilder::build_member<NestedBuilder>(id, stream_memento);
533     }
534
535     // Create a nested builder
536     UnionBuilder(nested_tag_t, AbstractBuilder& parent, unsigned int alignment)
537         : AggregationBuilder(nested_tag_t(), parent, alignment)
538     {
539     }
540
541 private:
542     void move_to_discriminator()
543     {
544         stream().current_position(begin_position() + RTI_XCDR_DHEADER_SIZE);
545     }
546
547     bool add_discriminator(Discriminator disc)
548     {
549         MemberParams params;
550         params.discriminator = true;
551         return AggregationBuilder::add_primitive_member(0, disc, params);
552     }
553 };
554
555
556 } }
557
558 #endif  // RTI_DDS_FLAT_AGGREGATIONBUILDERS_HPP_
```

10.10 Builder.hpp

```
1 /*
2 (c) Copyright, Real-Time Innovations, 2018.
3 All rights reserved.
4
5 No duplications, whole or partial, manual or electronic, may be made
6 without express written permission.  Any such copies, or
7 revisions thereof, must display this notice unaltered.
8 This code contains trade secrets of Real-Time Innovations, Inc.
9 */
10
11 #ifndef RTI_DDS_FLAT_BUILDER_HPP_
12 #define RTI_DDS_FLAT_BUILDER_HPP_
13
14 #include "xcdr/xcdr_stream.h"
15 #include "xcdr/xcdr_stream_impl.h"
16 #include "xcdr/xcdr_interpreter.h"
```

```
17 #include "xcdr/xcdr_interpreter.h"
18
19 #include "rti/xcdr/Stream.hpp"
20 #include "rti/flat/ExceptionHelper.hpp"
21 #include "rti/flat/SequenceOffsets.hpp"
22
124 #ifdef DOXYGEN_DOCUMENTATION_ONLY
125
206 class NDDSUSERDllExport MyFlatMutableBuilder : public rti::flat::AggregationBuilder {
207   public:
211     typedef MyFlatMutableOffset Offset;
212
222     MyFlatMutableBuilder()
223     {
224     }
225
242     MyFlatMutableBuilder(
243             unsigned char *buffer,
244             int32_t size,
245             bool initialize_members = false)
246     {
247     }
248
257     Offset finish();
258
274     MyFlatMutable * finish_sample();
275
279     bool add_my_primitive(int32_t value);
280
284     bool add_my_optional_primitive(int32_t value);
285
291     rti::flat::PrimitiveArrayOffset<int32_t, 10> add_my_primitive_array();
292
296     rti::flat::PrimitiveSequenceBuilder<int32_t> build_my_primitive_seq();
297
303     MyFlatFinalOffset add_my_final();
304
310     rti::flat::FinalAlignedArrayOffset<MyFlatFinalOffset, 10> add_my_final_array();
311
315     rti::flat::FinalSequenceBuilder<MyFlatFinalOffset> build_my_final_seq();
316
322     FlatMutableBarBuilder build_my_mutable();
323
327     rti::flat::MutableArrayBuilder<FlatMutableBarBuilder, 10> build_my_mutable_array();
328
332     rti::flat::MutableSequenceBuilder<FlatMutableBarBuilder > build_my_mutable_seq();
333
337     rti::flat::StringBuilder build_my_string();
338
342     rti::flat::MutableSequenceBuilder<rti::flat::StringBuilder > build_my_string_seq();
343 };
344
361 class MyFlatUnionBuilder : public rti::flat::UnionBuilder<int32_t> {
362 public:
366     typedef MyFlatUnionOffset Offset;
367
368
378     MyFlatUnionBuilder()
379     {
380     }
381
387     Offset finish();
388
389
395     MyFlatUnion * finish_sample();
396
403     bool add_my_primitive(int32_t value);
404
414     MyFlatMutableBuilder build_my_mutable(int32_t discriminator = 1);
415
424     MyFlatFinal::Offset add_my_final();
425 };
426
427 #endif
428
429 namespace rti { namespace flat {
430
431 // Support for move constructor and assignment operator in C++98
432 //
433 // All concrete subclasses of AbstractBuilder must use this macro to implement
434 // move semantics
```

```
435 //
436 #if defined(RTI_FLAT_DATA_CXX11_RVALUE_REFERENCES)
437 #define RTI_FLAT_BUILDER_DEFINE_MOVE_OPERATIONS_IMPL(TYPE, BASE, PROXY)
438 #define RTI_FLAT_BUILDER_DEFINE_MOVE_OPERATIONS(TYPE, BASE)
439 #define RTI_FLAT_MOVE_BUILDER(BUILDER) BUILDER
440 #else
441 #define RTI_FLAT_BUILDER_DEFINE_MOVE_OPERATIONS_IMPL(TYPE, BASE, PROXY) \
442 public: \
443     TYPE(PROXY other) throw() \
444     { \
445         this->move_from(other); \
446     } \
447     TYPE& operator=(PROXY other) throw() \
448     { \
449         this->finish_untyped_impl(); \
450         this->move_from(other); \
451         return *this; \
452     } \
453     TYPE move() \
454     { \
455         return TYPE(PROXY(*this)); \
456     } \
457 private: \
458     TYPE(TYPE&); \
459     TYPE& operator=(TYPE&);
460
461 // Shortcut for generated IDL types
462 #define RTI_FLAT_BUILDER_DEFINE_MOVE_OPERATIONS(TYPE, BASE) \
463     RTI_FLAT_BUILDER_DEFINE_MOVE_OPERATIONS_IMPL( \
464         TYPE, BASE, UntypedAggregationBuilderMoveProxy)
465
466 #define RTI_FLAT_MOVE_BUILDER(BUILDER) (BUILDER).move()
467 #endif
468
469 namespace detail {
470     // Template parameters can be AbstractBuilder or AbstractBuilderMoveProxy
471     template <typename Builder1, typename Builder2>
472     void move_abstract_builder(Builder1& to, Builder2& from)
473     {
474         if (from.parent_builder_ == NULL) {
475             to.owned_stream_ = from.owned_stream_;
476         }
477
478         to.parent_stream_ = from.parent_stream_;
479         to.parent_builder_ = from.parent_builder_;
480         to.begin_position_ = from.begin_position_;
481         to.bind_position_ = from.bind_position_;
482         to.initialize_on_add_ = from.initialize_on_add_;
483 #ifdef RTI_FLAT_DATA_NO_EXCEPTIONS
484         to.failure_ = from.failure_;
485 #endif
486
487         from.parent_stream_ = NULL;
488         from.parent_builder_ = NULL;
489         from.bind_position_ = NULL;
490         from.begin_position_ = NULL;
491     }
492 }
493
494 /*
495  * @brief Provides the functionality common to all builders
496  *
497  * This class is the base of all builders and provides the following
498  * functionality:
499  *
500  * - Get the buffer and its size
501  * - Add final elements (base case)
502  * - Build operations--create nested builders, binding (base case)
503  * - Finish operation--unbind (base case)
504  * - Discard operation--throw away a nested Builder and roll back the parent
505  * - Error management when exceptions are not supported (traditional C++)
506  * - Move semantics (base case)
507  */
512 class AbstractBuilder {
513 protected:
514     AbstractBuilder()
515         : parent_stream_(NULL),
516           parent_builder_(NULL),
517           begin_position_(NULL),
518           bind_position_(NULL),
519           initialize_on_add_(false)
```

```
520 #ifdef RTI_FLAT_DATA_NO_EXCEPTIONS
521           , failure_(false)
522 #endif
523     {
524     }
525 
526     /*i
527      * @brief Create a new top-level Builder
528      */
529     AbstractBuilder(unsigned char *buffer, offset_t size, bool initialize_members)
530         : parent_stream_(NULL),
531           parent_builder_(NULL),
532           bind_position_(NULL),
533           initialize_on_add_(initialize_members)
534 #ifdef RTI_FLAT_DATA_NO_EXCEPTIONS
535           , failure_(false)
536 #endif
537     {
538         // Serializes the encapsulation into the buffer and initializes the
539         // stream with that buffer and that encapsulation
540         if (!RTIXCdrFlatSample_initializeEncapsulationAndStream(
541                 (char *) buffer,
542                 &owned_stream_.c_stream(),
543                 RTIXCdrEncapsulationId_getNativePlCdr2(),
544                 size)) {
545             RTI_FLAT_BUILDER_OUT_OF_RESOURCES_ERROR(return);
546         }
547 
548         begin_position_ = owned_stream_.current_position();
549     }
550 
551 protected:
552     template <typename Builder1, typename Builder2>
553     friend void detail::move_abstract_builder(Builder1& to, Builder2& from);
554 
555 #if defined(RTI_FLAT_DATA_CXX11_RVALUE_REFERENCES)
556     AbstractBuilder(AbstractBuilder&& other)
557     {
558         detail::move_abstract_builder(*this, other);
559     }
560 
561     AbstractBuilder& operator=(AbstractBuilder&& other)
562     {
563         if (this == &other) {
564             return *this;
565         }
566 
567         finish_untyped_impl();
568 
569         detail::move_abstract_builder(*this, other);
570 
571         return *this;
572     }
573 #else
574     // Enables the safe-move-constructor idiom without C++11 move constructors
575     struct AbstractBuilderMoveProxy {
576         rti::xcdr::Stream owned_stream_;
577         rti::xcdr::Stream *parent_stream_;
578         AbstractBuilder *parent_builder_;
579         unsigned char *begin_position_;
580         unsigned char *bind_position_;
581         bool initialize_on_add_;
582 #ifdef RTI_FLAT_DATA_NO_EXCEPTIONS
583         bool failure_;
584 #endif
585     };
586 
587     void move_from(AbstractBuilderMoveProxy& other)
588     {
589         detail::move_abstract_builder(*this, other);
590     }
591 
592     void move_to(AbstractBuilderMoveProxy& other)
593     {
594         detail::move_abstract_builder(other, *this);
595     }
596 
597     operator AbstractBuilderMoveProxy () throw() // move-constructor idiom
598     {
599         AbstractBuilderMoveProxy other;
600         move_to(other);
```

```
601         return other;
602     }
603 private:
604     AbstractBuilder(AbstractBuilder& other);
605     AbstractBuilder& operator=(AbstractBuilder& other);
606 #endif
607
608 protected:
620     virtual ~AbstractBuilder()
621     {
622         finish_untyped_impl();
623     }
624
625     struct nested_tag_t {}; // disambiguate copy constructor
626
627     /*i
628      * @brief Creates a nested Builder to build a member or element
629      *
630      * @param parent The Builder for the type that contains the member or element
631      * this Builder builds.
632      *
633      * @param alignment Specifies the alignment required by the concrete Builder.
634      * Namely, aggregation and sequence builders require alignment of 4 for the
635      * DHeader and the sequence length, respectively. This alignment doesn't
636      * reflect the alignment requirement of the elements, that's why arrays do
637      * not require an alignment here. If no alignment is required, this parameter
638      * must be zero. Concrete classes must provide a default value for alignment,
639      * that indicates their requirement. This parameter can be overridden by
640      * build_element_no_align to when we know we don't need to align.
641      */
642     AbstractBuilder(
643             nested_tag_t,
644             AbstractBuilder& parent,
645             unsigned int alignment)
646         : /* owned_stream_ empty and unused */
647           parent_stream_(&parent.stream()), /* work on the parent's stream */
648           parent_builder_(&parent),
649           begin_position_(NULL),
650           bind_position_(NULL),
651           initialize_on_add_(parent.initialize_on_add_)
652 #ifdef RTI_FLAT_DATA_NO_EXCEPTIONS
653           , failure_(false)
654 #endif
655     {
656         if (alignment != 0) {
657             if (!stream().align(alignment)) {
658                 RTI_FLAT_BUILDER_OUT_OF_RESOURCES_ERROR(return);
659             }
660         }
661         begin_position_ = stream().current_position();
662     }
663
664     unsigned char * buffer()
665     {
666         return stream().buffer();
667     }
668
669     unsigned char * begin_position()
670     {
671         return begin_position_;
672     }
673
674     /*i
675      * @brief Adds a fixed-size member or element
676      *
677      * @return The offset to the member or element that was added. This offset
678      * can be used to initialize it.
679      */
680     template <typename OffsetType>
681     OffsetType add_element()
682     {
683         RTI_FLAT_BUILDER_CHECK_VALID(return OffsetType());
684         RTI_FLAT_BUILDER_CHECK_NOT_BOUND(return OffsetType());
685
686         offset_t member_size = OffsetType::serialized_size(0);
687         unsigned char *pos = stream().current_position();
688         if (detail::is_fixed_type_w_dheader<OffsetType>::value()) {
689             pos = (unsigned char *) stream().serialize_dheader();
690             if (pos == NULL) {
691                 RTI_FLAT_BUILDER_OUT_OF_RESOURCES_ERROR(return OffsetType());
692             }
```

```
693         }
694         if (!stream().skip(member_size)) {
695             RTI_FLAT_BUILDER_OUT_OF_RESOURCES_ERROR(return OffsetType());
696         }
697         if (detail::is_fixed_type_w_dheader<OffsetType>::value()) {
698             stream().finish_dheader((char *) pos);
699         }
700
701         if (initialize_on_add_) {
702             OffsetType offset(
703                 (SampleBase *) stream().buffer(),
704                 detail::ptrdiff(pos, stream().buffer()));
705
706             detail::final_offset_initializer<OffsetType>::initialize(offset);
707
708             return offset;
709         } else {
710             return OffsetType(
711                 (SampleBase *) stream().buffer(),
712                 detail::ptrdiff(pos, stream().buffer()));
713         }
714     }
715
716     /*i
717      * @brief Creates a nested Builder to build a variable-size member or element
718      *
719      * @param stream_memento Subclasses using build_element() to implement
720      * their own build method are required to provide a stream memento that
721      * contains the position in the stream before any bytes related to this
722      * member were added. Any error before the nested Builder is created will
723      * cause the memento's destructor to roll back the Builder to its state
724      * before this member was added. If the Builder can be created without errors
725      * stream_mement.discard() is called, and its position is kept in case
726      * the nested Builder is discarded (instead of finished), which also rolls
727      * back the parent Builder to its previous state.
728      *
729      * @post This Builder becomes "bound" until the returned Builder is finished.
730      * While bound, this Builder can't be used to add or build more elements.
731      *
732      * @return The Builder that allows building this member or element
733      */
734     template <typename NestedBuilder>
735     NestedBuilder build_element(rti::xcdr::Stream::Memento& stream_memento)
736     {
737         RTI_FLAT_BUILDER_CHECK_VALID(return NestedBuilder());
738         RTI_FLAT_BUILDER_CHECK_NOT_BOUND(return NestedBuilder());
739
740         NestedBuilder nested_builder(nested_tag_t(), *this);
741         RTI_FLAT_BUILDER_CHECK_CREATE_BUILDER(
742                 nested_builder,
743                 return NestedBuilder());
744
745         bind_position_ = stream_memento.discard();
746
747         // uninit_use_in_call is not possible given how the constructor for
748         // the NestedBuilder type is generated by rtiddsgen. The warning is
749         // suppressed for performance reasons.
750         /* coverity[uninit_use_in_call : FALSE] */
751         return RTI_FLAT_MOVE_BUILDER(nested_builder);
752     }
753
754     template <typename NestedBuilder>
755     NestedBuilder build_element_no_align(rti::xcdr::Stream::Memento& stream_memento)
756     {
757         RTI_FLAT_BUILDER_CHECK_VALID(return NestedBuilder());
758         RTI_FLAT_BUILDER_CHECK_NOT_BOUND(return NestedBuilder());
759
760         // By passing '0' to the builder constructor we override its default
761         // argument which indicates its required alignment
762         NestedBuilder nested_builder(nested_tag_t(), *this, 0);
763         RTI_FLAT_BUILDER_CHECK_CREATE_BUILDER(
764                 nested_builder,
765                 return NestedBuilder());
766
767         bind_position_ = stream_memento.discard();
768
769         // uninit_use_in_call is not possible given how the constructor for
770         // the NestedBuilder type is generated by rtiddsgen. The warning is
771         // suppressed for performance reasons.
772         /* coverity[uninit_use_in_call : FALSE] */
773         return RTI_FLAT_MOVE_BUILDER(nested_builder);
```

```
774     }
775
776     /*i
777      * @brief Returns the currently used number of bytes
778      *
779      * The current size is the number of bytes that have been used to create
780      * this sample by adding or building members.
781      *
782      * When current_size() reaches capacity(), the Builder will fail to add
783      * or build any additional member.
784      *
785      * @see capacity().
786      */
787     unsigned int current_size()
788     {
789         return detail::ptrdiff(stream().current_position(), begin_position_);
790     }
791
792
793     friend class AggregationBuilder; // to access finish_member()
794
795     // Finishes this nested builder, updating the parent builder.
796     //
797     // This function doesn't throw exceptions and doesn't return a typed Offset
798     // This function can be used in destructors to "silently" finish the builder
799     void finish_untyped_impl()
800     {
801         if (!is_valid() || !is_nested()) {
802             return;
803         }
804
805         parent_builder_->finish_member();
806         invalidate();
807     }
808
809 protected:
810     /*i
811      * @brief Complete the building of a nested member
812      *
813      * Concrete Builders must implement a finish() function with the following body:
814      *
815      * RTI_FLAT_BUILDER_CHECK_CAN_FINISH(...);
816      * // optionally, additonal work
817      * return finish_impl<OffsetType>();
818      *
819      * Destructors can directly call finish_untyped_impl();
820      *
821      * When this is a nested Builder, this function indicates that this element
822      * or member is complete.
823      *
824      * This function is automatically called by the Builder destructor.
825      *
826      * Concrete builders must override this function to return a typed Offset to
827      * the element that was built.
828      *
829      * @post This Builder is in an empty state and cannot be used further
830      *
831      * @return The position in the buffer where this element ended.
832      */
833     template <typename OffsetType>
834     OffsetType finish_impl()
835     {
836         RTI_FLAT_BUILDER_CHECK_VALID(return OffsetType());
837
838         unsigned char *begin_pos = this->begin_position_;
839         unsigned char *sample_base = stream().buffer();
840         unsigned char *current_pos = stream().current_position();
841         finish_untyped_impl();
842
843         return OffsetType(
844                 reinterpret_cast<SampleBase *>(sample_base), // start of the top-level sample
845                 detail::ptrdiff(begin_pos, sample_base), // absoute offset to the member
846                 detail::ptrdiff(current_pos, begin_pos)); // size of the member
847     }
848
849 public:
863     void discard()
864     {
865         RTI_FLAT_BUILDER_CHECK_VALID(return);
866         RTI_FLAT_BUILDER_CHECK_CAN_FINISH(return);
867
```

```
868         parent_builder_->discard_member();
869         invalidate();
870     }
871
882     bool is_nested() const
883     {
884         return parent_builder_ != NULL;
885     }
886
899     bool is_valid() const
900     {
901         return begin_position_ != NULL;
902     }
903
912     rti::xcdr::length_t capacity() const
913     {
914         return stream().total_size();
915     }
916
917 protected:
918     // Makes this Builder invalid after finish(); discard(); finish_sample();
919     // and, when exceptions are not enabled (traditional C++ API), after an
920     // error during construction
921     void invalidate()
922     {
923         parent_stream_ = NULL;
924         parent_builder_ = NULL;
925         bind_position_ = NULL;
926         begin_position_ = NULL;
927     }
928
929     // This function is called by a nested Builder when it has finished building
930     // an element
931     virtual void finish_member() // noexcept
932     {
933         if (!is_valid()) {
934             return;
935         }
936
937         bind_position_ = NULL;
938     }
939
940     void discard_member()
941     {
942         RTI_FLAT_BUILDER_CHECK_VALID(return);
943
944         stream().current_position(bind_position_);
945         bind_position_ = NULL;
946     }
947
948     rti::xcdr::Stream& stream()
949     {
950         if (parent_stream_ != NULL) {
951             return *parent_stream_;
952         } else {
953             return owned_stream_;
954         }
955     }
956
957     const rti::xcdr::Stream& stream() const
958     {
959         if (parent_stream_ != NULL) {
960             return *parent_stream_;
961         } else {
962             return owned_stream_;
963         }
964     }
965
966 private:
967     rti::xcdr::Stream owned_stream_;
968     rti::xcdr::Stream *parent_stream_;
969     AbstractBuilder *parent_builder_;
970     unsigned char *begin_position_;
971     unsigned char *bind_position_;
972     bool initialize_on_add_;
973
974 #ifdef RTI_FLAT_DATA_NO_EXCEPTIONS
975 public:
986     bool check_failure()
987     {
988         bool failure = failure_;
```

```
989         failure_ = false;
990         return failure;
991     }
992 protected:
993     void set_failure()
994     {
995         failure_ = true;
996     }
997 private:
998     bool failure_;
999 #endif
1000
1001
1002 };
1003
1004 } }
1005
1006 #endif // RTI_DDS_FLAT_BUILDER_HPP_
1007
```

10.11 BuilderHelper.hpp

```
1 /*
2 (c) Copyright, Real-Time Innovations, 2018.
3 All rights reserved.
4
5 No duplications, whole or partial, manual or electronic, may be made
6 without express written permission.  Any such copies, or
7 revisions thereof, must display this notice unaltered.
8 This code contains trade secrets of Real-Time Innovations, Inc.
9 */
10
11 #ifndef RTI_DDS_FLAT_BUILDERHELPER_HPP_
12 #define RTI_DDS_FLAT_BUILDERHELPER_HPP_
13
14 namespace rti { namespace flat {
15
16 template <typename T>
17 class PrimitiveSequenceBuilder;
18
19 template <typename ElementBuilder>
20 class MutableSequenceBuilder;
21
22 template <typename ElementOffset>
23 class FinalSequenceBuilder;
24
25 template <typename ElementBuilder, unsigned int N>
26 class MutableArrayBuilder;
27
28 class StringBuilder;
29
30 namespace detail {
31
32 //
33 // lc_code:
34 //
35
36 template <size_t PrimitiveSize>
37 struct primitive_lc_code_helper;
38
39 template <>
40 struct primitive_lc_code_helper<1> {
41     enum {
42         single = 0, // LC = 1 => member length is 1
43         sequence = 5 // LC = 5 => member length is also NEXTINT
44     };
45 };
46
47 template <>
48 struct primitive_lc_code_helper<2> {
49     enum {
50         single = 1, // LC = 1 => member length is 2
51         sequence = 4 // LC = 4 => member length is provided as an additional int
52     };
53 };
54
55 template <>
```

```
56 struct primitive_lc_code_helper<4> {
57     enum {
58         single = 2, // LC = 2 => member length is 4
59         sequence = 6 // LC = 6 => member length is 4 * NEXTINT
60     };
61 };
62
63 template <>
64 struct primitive_lc_code_helper<8> {
65     enum {
66         single = 3, // LC = 3 => member length is 8
67         sequence = 7 // LC = 7 => member length is 8 * NEXTINT
68     };
69 };
70
71 template <typename T>
72 struct primitive_lc_code {
73     enum { value = primitive_lc_code_helper<sizeof(T)>::single };
74 };
75
76 // In general, use LC = 5, for mutable aggregation types. Final aggregation
77 // types don't use this 'lc_code' trait type. The other cases are specialized
78 // after this.
79 template <typename T, typename Enable = void>
80 struct lc_code {
81     static int value()
82     {
83         return 5;  // LC = 5 => member length is provided in NEXTINT/
84     }
85 };
86
87 // For collections of complex types and enums, if we serialize the DHeader
88 // (XCDR2 compliant behavior) we use LC = 5; if we don't (legacy RTI behavior)
89 // we use LC = 4, since we codify the length in the member header.
90 template <typename T>
91 struct lc_code<MutableSequenceBuilder<T> > {
92     static int value()
93     {
94         return is_dheader_required_in_non_primitive_collections() ? 5 : 4;
95     }
96 };
97
98 template <typename T>
99 struct lc_code<FinalSequenceBuilder<T> > {
100     static int value()
101     {
102         return is_dheader_required_in_non_primitive_collections() ? 5 : 4;
103     }
104 };
105
106 template <typename T, unsigned int N>
107 struct lc_code<MutableArrayBuilder<T, N> > {
108     static int value()
109     {
110         return is_dheader_required_in_non_primitive_collections() ? 5 : 4;
111     }
112 };
113
114 #if defined(RTI_FLAT_DATA_NO_EXCEPTIONS)
115
116 // Traditional C++: no support for std::is_enum<T>
117
118 template <typename T>
119 struct lc_code<PrimitiveSequenceBuilder<T> > {
120     static int value()
121     {
122         return primitive_lc_code_helper<sizeof(T)>::sequence;
123     }
124 };
125
126 #else
127
128 template <typename T>
129 struct lc_code<
130         PrimitiveSequenceBuilder<T>,
131         typename std::enable_if<!std::is_enum<T>::value>::type> {
132     static int value()
133     {
134         return primitive_lc_code_helper<sizeof(T)>::sequence;
135     }
136 };
```

```
137
138 template <typename T>
139 struct lc_code<
140         PrimitiveSequenceBuilder<T>,
141         typename std::enable_if<std::is_enum<T>::value>::type> {
142     static int value()
143     {
144         return is_dheader_required_in_enum_collections()
145                 ? 5
146                 : primitive_lc_code_helper<sizeof(T)>::sequence;
147     }
148 };
149
150 #endif
151
152
153 //
154 // Initialization of final types
155 //
156
157 template <typename Offset>
158 struct final_offset_initializer {
159
160     static bool initialize(Offset& offset)
161     {
162         return RTI_XCDR_TRUE == RTIXCdrFlatData_initializeSample(
163                 (char *) offset.get_buffer(),
164                 offset.get_buffer_size(),
165                 rti::xcdr::type_programs<rti::flat::Sample<Offset> >::get());
166     }
167
168 };
169
170 // This will be specialized for enums in generated code (traditional C++)
171 // and for enum classes in FlatDataTraits.hpp (modern C++)
172 template <typename T, typename Enable = void>
173 struct default_primitive_value {
174     static T get()
175     {
176         return (T) 0;
177     }
178 };
179
180 template <typename T, unsigned int N>
181 struct final_offset_initializer<rti::flat::PrimitiveArrayOffset<T, N> > {
182
183     static bool initialize(rti::flat::PrimitiveArrayOffset<T, N>& array)
184     {
185         T default_value = default_primitive_value<T>::get();
186         if (default_value == (T) 0) { // don't need to deal with endianness
187             memset(array.get_buffer(), 0, array.get_buffer_size());
188         } else {
189             for (unsigned int i = 0; i < array.element_count(); i++) {
190                 array.set_element(i, default_value);
191             }
192         }
193         return true;
194     }
195 };
196
197 template <typename T, unsigned int N>
198 struct final_offset_initializer<rti::flat::FinalAlignedArrayOffset<T, N> > {
199
200     static bool initialize(rti::flat::FinalAlignedArrayOffset<T, N>& array)
201     {
202         for (unsigned int i = 0; i < N; i++) {
203             T offset = array.get_element(i);
204             if (offset.is_null()) {
205                 return false;
206             }
207
208             if (!final_offset_initializer<T>::initialize(offset)) {
209                 return false;
210             }
211         }
212
213         return true;
214     }
215 };
216
217 } } }
```

```
218
219 #endif // RTI_DDS_FLAT_BUILDERHELPER_HPP_
220
```

10.12 ExceptionHelper.hpp

```
1 /*
2 (c) Copyright, Real-Time Innovations, 2018.
3 All rights reserved.
4
5 No duplications, whole or partial, manual or electronic, may be made
6 without express written permission.  Any such copies, or
7 revisions thereof, must display this notice unaltered.
8 This code contains trade secrets of Real-Time Innovations, Inc.
9 */
10
11
12 #ifndef RTI_DDS_FLAT_EXCEPTIONHELPER_HPP_
13 #define RTI_DDS_FLAT_EXCEPTIONHELPER_HPP_
14
15 //
16 // Enable debug assertions (RTI_FLAT_ASSERT) in debug mode
17 //
18 #if !defined(NDEBUG)
19 #define RTI_FLAT_DATA_ENABLE_DEBUG_ASSERTIONS
20 #endif
21
22 //
23 // Error handling without C++ exceptions (traditional C++ and Micro C++ APIs)
24 //
25 #ifdef RTI_FLAT_DATA_NO_EXCEPTIONS
26     // TODO: use MSG
27     #ifdef RTI_FLAT_DATA_ENABLE_DEBUG_ASSERTIONS
28         #define RTI_FLAT_ASSERT(COND, ACTION) \
29             if (!(COND)) { \
30                 RTIXCdrFlatData_logPreconditionError(__FILE__, __LINE__, "Assertion failure: " #COND); \
31                 ACTION; \
32             }
33     #else
34         #define RTI_FLAT_ASSERT(COND, ACTION)
35     #endif
36
37     // CR decission: Log error messages. Export function for each log kind (serialize, deserialize,
       precondition, skip)
38     // pass type/field (if possible) to serialization errors
39     #define RTI_FLAT_OFFSET_CHECK_NOT_NULL(ACTION) \
40         if (this->is_null()) { \
41             RTIXCdrFlatData_logPreconditionError(__FILE__, __LINE__, "null offset"); \
42             ACTION; \
43         }
44
45     #define RTI_FLAT_OFFSET_PRECONDITION_ERROR(MSG, ACTION) \
46             RTIXCdrFlatData_logPreconditionError(__FILE__, __LINE__, MSG); \
47             ACTION
48
49     #define RTI_FLAT_CHECK_PRECONDITION(COND, ACTION) \
50         if (!(COND))  { \
51             RTIXCdrFlatData_logPreconditionError(__FILE__, __LINE__, "Precondition failure: " #COND); \
52             ACTION; \
53         }
54
55     #define RTI_FLAT_BUILDER_CHECK_NOT_BOUND(ACTION) \
56         if (this->bind_position_)  { \
57             RTIXCdrFlatData_logPreconditionError(__FILE__, __LINE__, "Builder is currently bound - call
       finish() first"); \
58             ACTION; \
59         }
60
61     #define RTI_FLAT_BUILDER_CHECK_CAN_FINISH(ACTION) \
62         if (!this->is_nested())  { \
63             RTIXCdrFlatData_logPreconditionError(__FILE__, __LINE__, "This is not a member builder; call
       finish_sample() instead"); \
64             ACTION; \
65         }
66
67     #define RTI_FLAT_BUILDER_CHECK_CAN_FINISH_SAMPLE(ACTION) \
68         if (this->is_nested())  { \
```

```
69              RTIXCdrFlatData_logPreconditionError(__FILE__, __LINE__, "This is a member builder; call finish()
    instead"); \
70              ACTION; \
71          }
72
73      #define RTI_FLAT_BUILDER_CHECK_VALID(ACTION) \
74          if (!this->is_valid()) { \
75              RTIXCdrFlatData_logPreconditionError(__FILE__, __LINE__, "This Builder is not valid"); \
76              ACTION; \
77          }
78
79      #define RTI_FLAT_BUILDER_CHECK_CREATE_BUILDER(BUILDER, ACTION) \
80          if (!BUILDER.is_valid()) { \
81              this->set_failure(); \
82              RTIXCdrFlatData_logCreationError(__FILE__, __LINE__, "Builder"); \
83              ACTION; \
84          }
85
86      #define RTI_FLAT_BUILDER_OUT_OF_RESOURCES_ERROR(ACTION) \
87          this->set_failure(); \
88          RTIXCdrFlatData_logBuilderOutOfResources(__FILE__, __LINE__); \
89          ACTION
90
91      #define RTI_FLAT_BUILDER_PRECONDITION_ERROR(MSG, ACTION) \
92          this->set_failure(); \
93          RTIXCdrFlatData_logPreconditionError(__FILE__, __LINE__, MSG); \
94          ACTION
95
96  //
97  // Error handling with C++ exceptions (modern C++ API)
98  //
99  #else // !RTI_FLAT_DATA_NO_EXCEPTIONS
100
101     #ifdef RTI_FLAT_DATA_ENABLE_DEBUG_ASSERTIONS
102         #define RTI_FLAT_ASSERT(COND, ACTION) \
103             if (!(COND)) throw dds::core::PreconditionNotMetError("Assertion failure: " #COND)
104     #else
105         #define RTI_FLAT_ASSERT(COND, ACTION)
106     #endif
107
108     #define RTI_FLAT_OFFSET_CHECK_NOT_NULL(ACTION) \
109         if (this->is_null()) throw dds::core::NullReferenceError("null offset")
110
111     #define RTI_FLAT_OFFSET_PRECONDITION_ERROR(MSG, ACTION) \
112             throw dds::core::PreconditionNotMetError(MSG)
113
114     #define RTI_FLAT_CHECK_PRECONDITION(COND, ACTION) \
115         if (!(COND)) throw dds::core::PreconditionNotMetError("Precondition failure: " #COND)
116
117     #define RTI_FLAT_BUILDER_CHECK_NOT_BOUND(ACTION) \
118         if (bind_position_) \
119             throw dds::core::PreconditionNotMetError( \
120                 "Builder is currently bound - call finish() first")
121
122     #define RTI_FLAT_BUILDER_CHECK_CAN_FINISH(ACTION) \
123         if (!is_nested()) \
124             throw dds::core::PreconditionNotMetError( \
125                 "This is not a member builder; call finish_sample() instead")
126
127     #define RTI_FLAT_BUILDER_CHECK_CAN_FINISH_SAMPLE(ACTION) \
128         if (is_nested()) \
129             throw dds::core::PreconditionNotMetError( \
130                 "This is a member builder; call finish() instead")
131
132     #define RTI_FLAT_BUILDER_CHECK_VALID(ACTION) \
133         if (!is_valid()) \
134             throw dds::core::PreconditionNotMetError("This Builder is not valid")
135
136     // With exceptions, nothing to check (exception would have been thrown)
137     #define RTI_FLAT_BUILDER_CHECK_CREATE_BUILDER(BUILDER, ACTION)
138
139     #define RTI_FLAT_BUILDER_OUT_OF_RESOURCES_ERROR(ACTION) \
140         throw dds::core::OutOfResourcesError("builder reached end of buffer")
141
142     #define RTI_FLAT_BUILDER_PRECONDITION_ERROR(MSG, ACTION) \
143         throw dds::core::PreconditionNotMetError(MSG)
144
145
146 #endif // RTI_FLAT_DATA_NO_EXCEPTIONS
147
148 //
```

```
149 // Error checking (other than assertions) in release mode is enabled unless
150 // the sources are compiled with -DRTI_FLAT_DATA_DISABLE_ERROR_CHECKING
151 //
152 // In that case, most error checks that deal with preconditions due to incorrect
153 // usage of the API are disabled in release mode. This would lead to undefined
154 // behavior in case of bad usage, but can improve performance.
155 //
156 #if defined(RTI_FLAT_DATA_ENABLE_DEBUG_ASSERTIONS) \
157     || !defined(RTI_FLAT_DATA_DISABLE_ERROR_CHECKING)
158 #define RTI_FLAT_DATA_ENABLE_ERROR_CHECKING
159 #endif
160
161 #ifndef RTI_FLAT_DATA_ENABLE_ERROR_CHECKING
162
163     #define RTI_FLAT_OFFSET_CHECK_NOT_NULL(ACTION)
164     #define RTI_FLAT_ASSERT(COND, ACTION)
165     #define RTI_FLAT_CHECK_PRECONDITION(COND, ACTION)
166     #define RTI_FLAT_BUILDER_CHECK_NOT_BOUND(ACTION)
167     #define RTI_FLAT_BUILDER_CHECK_CAN_FINISH(ACTION)
168     #define RTI_FLAT_BUILDER_CHECK_CAN_FINISH_SAMPLE(ACTION)
169     #define RTI_FLAT_BUILDER_CHECK_VALID(ACTION)
170     #define RTI_FLAT_BUILDER_CHECK_CREATE_BUILDER(BUILDER, ACTION)
171
172 #endif // RTI_FLAT_DATA_ENABLE_ERROR_CHECKING
173
174 namespace rti { namespace flat { namespace detail {
175
176 } } }
177
178 #endif // RTI_DDS_FLAT_EXCEPTIONHELPER_HPP_
179
```

10.13 FlatSample.hpp

```
1 /*
2 (c) Copyright, Real-Time Innovations, 2018.
3 All rights reserved.
4
5 No duplications, whole or partial, manual or electronic, may be made
6 without express written permission.  Any such copies, or
7 revisions thereof, must display this notice unaltered.
8 This code contains trade secrets of Real-Time Innovations, Inc.
9 */
10
11 #ifndef RTI_DDS_FLAT_FLATSAMPLE_HPP_
12 #define RTI_DDS_FLAT_FLATSAMPLE_HPP_
13
14 // Note: FlatSampleImpl.hpp needs to be included in addition to this header.
15
16 #include <cstdlib>
17 #include <memory>
18
19 #include "xcdr/xcdr_stream_impl.h"
20 #include "xcdr/xcdr_flat_data.h"
21 #include "xcdr/xcdr_interpreter.h"
22
23 #include "rti/xcdr/Interpreter.hpp" // for type_programs
24
39 namespace rti { namespace flat {
40
41 struct fixed_size_type_tag_t {};
42 struct variable_size_type_tag_t {};
43
44 typedef RTIXCdrUnsignedLong offset_t;
45 typedef RTIXCdrUnsignedLong member_id_t;
46
47 // The untemplated base of Sample<T>
48 class SampleBase {
49 public:
50
51     unsigned char* get_buffer()
52     {
53         return &data[0];
54     }
55
56     const unsigned char* get_buffer() const
57     {
```

```
58          return &data[0];
59      }
60
61      unsigned char* get_root_buffer()
62      {
63          return get_buffer() + RTI_XCDR_ENCAPSULATION_SIZE;
64      }
65
66      /*i
67       * @brief Provides the address of the top-level type
68       */
69      const unsigned char* get_root_buffer() const
70      {
71          return get_buffer() + RTI_XCDR_ENCAPSULATION_SIZE;
72      }
73
74      void initialize_stream(
75          RTIXCdrStream& stream,
76          offset_t offset,
77          offset_t serialized_size) const
78      {
79          RTIXCdrFlatData_initializeStream(
80                  &stream,
81                  const_cast<unsigned char*>(get_buffer()),
82                  offset,
83                  serialized_size);
84      }
85
86  protected: // xlc compiler fails when the default constructor is private
87  #if !defined(RTI_FLAT_DATA_CXX11_DELETED_FUNCTIONS)
88      SampleBase() {} // Samples can only be created via Sample::create_data
89      SampleBase(const SampleBase&) {}
90      SampleBase& operator=(const SampleBase&);
91  #else
92      SampleBase() = delete;
93      SampleBase(const SampleBase&) = delete;
94      SampleBase& operator=(const SampleBase&) = delete;
95  #endif
96
97      // The actual size of data (and therefore, the Sample object) is variable,
98      // and equal to the serialized size of the IDL Sample
99      unsigned char data[1];
100 };
101
147 template <typename OffsetType>
148 class Sample : public SampleBase {
149 public:
155     typedef OffsetType Offset;
156
162     typedef typename OffsetType::ConstOffset ConstOffset;
163
181     Offset root()
182     {
183         return root_impl(typename Offset::offset_kind());
184     }
185
203     ConstOffset root() const
204     {
205         return root_impl(typename ConstOffset::offset_kind());
206     }
207
211     RTIXCdrEndian endian() const
212     {
213         RTIXCdrStream tmp_stream;
214         initialize_stream(tmp_stream, 0, RTI_XCDR_ENCAPSULATION_SIZE);
215         return tmp_stream._endian;
216     }
217
221     static Sample<OffsetType>* from_buffer(unsigned char *buffer)
222     {
223         return reinterpret_cast<Sample<OffsetType>*>(buffer);
224     }
225
245     static Sample<OffsetType>* create_data(); // impl in FlatSampleImpl.hpp
246
259     Sample<OffsetType>* clone() const
260     {
261         unsigned char *buffer = RTIXCdrFlatData_cloneSample(
262                 get_buffer(),
263                 sample_size());
264
```

```
265         if (buffer == NULL) {
266           #ifndef RTI_FLAT_DATA_NO_EXCEPTIONS
267             throw std::bad_alloc();
268           #else
269             return NULL;
270           #endif
271         }
272 
273         return reinterpret_cast<rti::flat::Sample<OffsetType> *>(buffer);
274     }
275 
286     static void delete_data(rti::flat::Sample<OffsetType>* sample)
287     {
288         RTIXCdrFlatData_deleteSample(sample);
289     }
290 
293     offset_t buffer_size() const
294     {
295         return sample_size() + RTI_XCDR_ENCAPSULATION_SIZE;
296     }
297 
300     offset_t sample_size() const
301     {
302         return sample_size_impl(typename OffsetType::offset_kind());
303     }
304 
305 private:
306     offset_t sample_size_impl(fixed_size_type_tag_t) const
307     {
308         return OffsetType::serialized_size(0);
309     }
310 
311     offset_t sample_size_impl(variable_size_type_tag_t) const
312     {
313         return RTIXCdrFlatSample_getMutableSampleSize(
314                 get_buffer(),
315                 RTI_XCDR_ENCAPSULATION_SIZE);
316     }
317 
318     Offset root_impl(fixed_size_type_tag_t)
319     {
320         return Offset(this, RTI_XCDR_ENCAPSULATION_SIZE);
321     }
322 
323     Offset root_impl(variable_size_type_tag_t)
324     {
325         return Offset(this, RTI_XCDR_ENCAPSULATION_SIZE, sample_size());
326     }
327 
328     ConstOffset root_impl(fixed_size_type_tag_t) const
329     {
330         return ConstOffset(this, RTI_XCDR_ENCAPSULATION_SIZE);
331     }
332 
333     ConstOffset root_impl(variable_size_type_tag_t) const
334     {
335         return ConstOffset(this, RTI_XCDR_ENCAPSULATION_SIZE, sample_size());
336     }
337 };
338 
339 template <typename T>
340 struct flat_type_traits;
341 
342 } }
343 
344 #ifdef DOXYGEN_DOCUMENTATION_ONLY
345 
375 typedef rti::flat::Sample<MyFlatFinalOffset> MyFlatFinal;
376 
416 typedef rti::flat::Sample<MyFlatMutableOffset> MyFlatMutable;
417 
447 typedef rti::flat::Sample<MyFlatUnionOffset> MyFlatUnion;
448 #endif
449 
450 #endif // RTI_DDS_FLAT_FLATSAMPLE_HPP_
451 
```

10.14 FlatSampleImpl.hpp

```
1 /*
2 (c) Copyright, Real-Time Innovations, 2018.
3 All rights reserved.
4
5 No duplications, whole or partial, manual or electronic, may be made
6 without express written permission.  Any such copies, or
7 revisions thereof, must display this notice unaltered.
8 This code contains trade secrets of Real-Time Innovations, Inc.
9 */
10
11 #ifndef RTI_DDS_FLAT_FLATSAMPLEIMPL_HPP_
12 #define RTI_DDS_FLAT_FLATSAMPLEIMPL_HPP_
13
14 // This file includes the implementation of some templates that need to
15 // appear after some types are defined in the generated code. This is a
16 // a restriction on some platforms such as pentiumInty11.pcx64 and other Intys,
17 // sparcSol2.10gcc3.4.2.
18
19 // For example, type_programs<T> needs to be defined (as an specialization
20 // for T) before its used in create_data().
21
22 #include "rti/flat/FlatSample.hpp"
23
24 namespace rti { namespace flat {
25
26 namespace detail {
27
28 template <typename OffsetType>
29 Sample<OffsetType>* create_data_impl(fixed_size_type_tag_t)
30 {
31     unsigned char *buffer = RTIXCdrFlatData_createSampleFinal(
32             OffsetType::serialized_size(0),
33             rti::xcdr::type_programs<Sample<OffsetType> >::get());
34
35     if (buffer == NULL) {
36         #ifndef RTI_FLAT_DATA_NO_EXCEPTIONS
37         throw std::bad_alloc();
38         #else
39         return NULL;
40         #endif
41     }
42
43     return reinterpret_cast<rti::flat::Sample<OffsetType> *>(buffer);
44 }
45
46 template <typename OffsetType>
47 Sample<OffsetType>* create_data_impl(variable_size_type_tag_t)
48 {
49     unsigned char *buffer = RTIXCdrFlatData_createSampleMutable(
50             rti::xcdr::type_programs<Sample<OffsetType> >::get());
51
52     if (buffer == NULL) {
53         #ifndef RTI_FLAT_DATA_NO_EXCEPTIONS
54         throw std::bad_alloc();
55         #else
56         return NULL;
57         #endif
58     }
59
60     return reinterpret_cast<rti::flat::Sample<OffsetType> *>(buffer);
61 }
62
63 }
64
65 // Declared in FlatSample.hpp
66 template <typename OffsetType>
67 Sample<OffsetType>* Sample<OffsetType>::create_data()
68 {
69     return detail::create_data_impl<OffsetType>(
70             typename OffsetType::offset_kind());
71 }
72
73 } }
74
75 #endif // RTI_DDS_FLAT_FLATSAMPLEIMPL_HPP_
76
```

10.15 FlatTypeTraits.hpp

```
1 /*
2 (c) Copyright, Real-Time Innovations, 2018.
3 All rights reserved.
4
5 No duplications, whole or partial, manual or electronic, may be made
6 without express written permission.  Any such copies, or
7 revisions thereof, must display this notice unaltered.
8 This code contains trade secrets of Real-Time Innovations, Inc.
9 */
10
11 #ifndef RTI_DDS_FLAT_FLATTYPETRAITS_HPP_
12 #define RTI_DDS_FLAT_FLATTYPETRAITS_HPP_
13
14 #include "rti/flat/Offset.hpp"
15 #include "rti/flat/SequenceOffsets.hpp"
16 #include "rti/flat/AggregationBuilders.hpp"
17
18 namespace rti { namespace flat {
19
20
64 // For aggregation types, this template is to be specialized in generated code
65 template <typename T>
66 struct flat_type_traits;
67
68 template <typename T, unsigned int N>
69 struct flat_type_traits<FinalArrayOffset<T, N> > {
70     typedef typename flat_type_traits<T>::plain_type plain_type;
71 };
72
73 template <typename T, unsigned int N>
74 struct flat_type_traits<PrimitiveArrayOffset<T, N> > {
75     typedef T plain_type;
76 };
77
78 template <typename T>
79 struct flat_type_traits<PrimitiveSequenceOffset<T> > {
80     typedef T plain_type;
81 };
82
83 template <typename T, unsigned int N>
84 struct flat_type_traits<FinalAlignedArrayOffset<T, N> > {
85     typedef typename flat_type_traits<T>::plain_type plain_type;
86 };
87
88 template <typename T, unsigned int N>
89 struct flat_type_traits<MutableArrayOffset<T, N> > {
90     typedef typename flat_type_traits<T>::plain_type plain_type;
91 };
92
93 template <typename T>
94 struct flat_type_traits<SequenceOffset<T> > {
95     typedef typename flat_type_traits<T>::plain_type plain_type;
96 };
97
98 template <>
99 struct flat_type_traits<StringOffset > :
100         flat_type_traits<PrimitiveSequenceOffset<char> > {
101 };
102
103 template <typename T>
104 struct flat_type_traits<PrimitiveOffset<T> > {
105     typedef T plain_type;
106 };
107
219 template <typename OffsetType>
220 typename flat_type_traits<OffsetType>::plain_type* plain_cast(OffsetType& offset)
221 {
222     RTI_FLAT_CHECK_PRECONDITION(offset.is_cpp_compatible(), return NULL);
223     return reinterpret_cast<typename flat_type_traits<OffsetType>::plain_type*>(
224             offset.get_buffer());
225 }
226
231 template <typename OffsetType>
232 const typename flat_type_traits<OffsetType>::plain_type* plain_cast(const OffsetType& offset)
233 {
234     RTI_FLAT_CHECK_PRECONDITION(offset.is_cpp_compatible(), return NULL);
235     return reinterpret_cast<const typename flat_type_traits<OffsetType>::plain_type*>(
236             offset.get_buffer());
```

```
237 }
238
239 namespace detail {
240
241 template <typename OffsetKind>
242 struct offset_kind_is_fixed_size {
243     enum { value = 0 };
244 };
245
246 template <>
247 struct offset_kind_is_fixed_size<rti::flat::fixed_size_type_tag_t> {
248     enum { value = 1 };
249 };
250
251 }
252
253 // Utility to determine if a Flat Data type is fixed size (final) or variable
254 // size (mutable)
255 template <typename T>
256 struct is_fixed_size_type
257         : detail::offset_kind_is_fixed_size<typename T::Offset::offset_kind>
258 {
259 };
260
261 } }
262
263 #endif // RTI_DDS_FLAT_FLATSAMPLE_HPP_
264
```

10.16 Offset.hpp

```
1 /*
2 (c) Copyright, Real-Time Innovations, 2018.
3 All rights reserved.
4
5 No duplications, whole or partial, manual or electronic, may be made
6 without express written permission.  Any such copies, or
7 revisions thereof, must display this notice unaltered.
8 This code contains trade secrets of Real-Time Innovations, Inc.
9 */
10
11 #ifndef RTI_DDS_FLAT_FLATOFFSETS_HPP_
12 #define RTI_DDS_FLAT_FLATOFFSETS_HPP_
13
14 #include "xcdr/xcdr_stream.h"
15 #include "xcdr/xcdr_stream_impl.h"
16 #include "xcdr/xcdr_interpreter.h"
17
18 #include "rti/xcdr/Stream.hpp"
19
108 #ifdef DOXYGEN_DOCUMENTATION_ONLY
109
125 class MyFlatFinalOffset : public rti::flat::FinalOffset<MyFlatFinalOffset> {
126   public:
135     typedef MyFlatFinalConstOffset ConstOffset;
136
142     MyFlatFinalOffset()
143     {
144     }
145
149     int32_t my_primitive() const;
150
156     FlatFinalBar::ConstOffset my_complex() const;
157
161     const rti::flat::PrimitiveArrayOffset<int32_t, 10> my_primitive_array() const;
162
166     rti::flat::FinalArrayOffset<FlatFinalBar::ConstOffset, 10> my_complex_array() const;
167
171     bool my_primitive(int32_t value);
172
178     FlatFinalBar::Offset my_complex();
179
183     rti::flat::PrimitiveArrayOffset<int32_t, 10> my_primitive_array();
184
188     rti::flat::FinalArrayOffset<FlatFinalBar::Offset, 10> my_complex_array();
189 };
190
```

```
203 class NDDSUSERDllExport MyFlatMutableOffset : public rti::flat::MutableOffset {
204   public:
213     typedef MyFlatMutableConstOffset ConstOffset;
214
220     MyFlatMutableOffset()
221     {
222     }
223
230     int32_t my_primitive() const;
231
240     rti::flat::PrimitiveConstOffset<int32_t> my_optional_primitive() const;
241
245     const rti::flat::PrimitiveArrayOffset<int32_t, 10> my_primitive_array() const;
246
250     const rti::flat::PrimitiveSequenceOffset<int32_t> my_primitive_seq() const;
251
255     MyFlatFinal::ConstOffset my_final() const;
256
260     rti::flat::FinalAlignedArrayOffset<MyFlatFinal::ConstOffset, 10> my_final_array() const;
261
265     rti::flat::SequenceOffset<MyFlatFinal::ConstOffset> my_final_seq() const;
266
272     FlatMutableBar::ConstOffset my_mutable() const;
273
277     rti::flat::MutableArrayOffset<FlatMutableBar::ConstOffset, 10> my_mutable_array() const;
278
282     rti::flat::SequenceOffset<FlatMutableBar::ConstOffset> my_mutable_seq() const;
283
287     const rti::flat::StringOffset my_string() const;
288
292     rti::flat::SequenceOffset<const rti::flat::StringOffset> my_string_seq() const;
293
297     bool my_primitive(int32_t value);
298
299     /* @brief Retrieves a non-const Offset to an optional primitive
300      *
301      * Unlike the non-optional my_primitive(), which allows accessing the integer
302      * directly, an optional primitive may not exist. If it doesn't, the Offset
303      * this function returns will be null (\p is_null()).
304      */
305     rti::flat::PrimitiveOffset<int32_t> my_optional_primitive();
306
310     rti::flat::PrimitiveArrayOffset<int32_t, 10> my_primitive_array();
311
315     rti::flat::PrimitiveSequenceOffset<int32_t> my_primitive_seq();
316
320     MyFlatFinal::Offset my_final();
321
325     rti::flat::FinalAlignedArrayOffset<MyFlatFinal::Offset, 10> my_final_array();
326
330     rti::flat::SequenceOffset<MyFlatFinal::Offset> my_final_seq();
331
337     FlatMutableBar::Offset my_mutable();
338
342     rti::flat::MutableArrayOffset<FlatMutableBar::Offset, 10> my_mutable_array();
343
347     rti::flat::SequenceOffset<FlatMutableBar::Offset> my_mutable_seq();
348
352     rti::flat::StringOffset my_string();
353
357     rti::flat::SequenceOffset<rti::flat::StringOffset> my_string_seq();
358 };
359
375 class NDDSUSERDllExport MyFlatUnionOffset
376 : public rti::flat::MutableOffset {
377   public:
381     typedef MyFlatUnionConstOffset ConstOffset;
382
388     MyFlatUnionOffset()
389     {
390     }
391
411     int32_t _d() const;
412
419     int32_t my_primitive() const;
420
427     MyFlatMutable::ConstOffset my_mutable() const;
428
435     MyFlatFinal::ConstOffset my_final() const;
436
445     bool my_primitive(int32_t value);
```

```
446
453     MyFlatMutable::Offset my_mutable();
454
461     MyFlatFinal::Offset my_final();
462 };
463
464 #endif
465
466 namespace rti { namespace flat {
467
468 namespace detail {
469
470 inline rti::flat::offset_t ptrdiff(unsigned char *a, unsigned char *b)
471 {
472     RTI_FLAT_ASSERT(a - b < RTI_XCDR_MAX_SERIALIZED_SIZE, return 0);
473
474     return static_cast<rti::flat::offset_t>(a - b);
475 }
476
477 // Used to determine which fixed-size types need a DHeader when adding a
478 // fixed-size member or element. OffsetType classes may specialize this template
479 // to indicate if they need a DHeader.
480 template <typename OffsetType, typename Enable = void>
481 struct is_fixed_type_w_dheader {
482     static bool value()
483     {
484         return false;
485     }
486 };
487
488 inline bool is_dheader_required_in_non_primitive_collections()
489 {
490     // Allow setting the behavior at compilation time to gain compiler
491     // optimizations
492 #if defined(RTI_FLATDATA_NO_DHEADER_COLLECTIONS)
493     return false;
494 #elif defined(RTI_FLATDATA_DHEADER_COLLECTIONS)
495     return true;
496 #else
497     // Without any compilation flag, the behavior is configurable at runtime
498     RTIXCdrXTypesComplianceMask xTypesComplianceMask;
499     xTypesComplianceMask = RTIXCdrInterpreter_getGlobalXtypeComplianceMask();
500     return RTIXCdrXTypesComplianceMask_isBitSet(
501             xTypesComplianceMask,
502             RTI_XCDR_XTYPES_DHEADER_IN_NON_PRIMITIVE_COLLECTIONS_BIT);
503 #endif
504 }
505
506 inline bool is_dheader_required_in_enum_collections()
507 {
508 #if defined(RTI_FLATDATA_NO_DHEADER_COLLECTIONS)
509     return false;
510 #elif defined(RTI_FLATDATA_DHEADER_COLLECTIONS)
511     return true;
512 #else
513     RTIXCdrXTypesComplianceMask xTypesComplianceMask;
514     xTypesComplianceMask = RTIXCdrInterpreter_getGlobalXtypeComplianceMask();
515     return RTIXCdrXTypesComplianceMask_isBitSet(
516             xTypesComplianceMask,
517             RTI_XCDR_XTYPES_DHEADER_IN_NON_PRIMITIVE_COLLECTIONS_BIT)
518             && !RTIXCdrXTypesComplianceMask_isBitSet(
519                     xTypesComplianceMask,
520                     RTI_XCDR_XTYPES_ENUM_AS_PRIMITIVE_IN_COLLECTIONS_BIT);
521 #endif
522 }
523
524 template <bool IsForEnum>
525 struct collection_dheader_traits {
526     static bool dheader_required()
527     {
528         return is_dheader_required_in_non_primitive_collections();
529     }
530
531     static unsigned int dheader_size()
532     {
533         return dheader_required() ? RTI_XCDR_DHEADER_SIZE : 0;
534     }
535 };
536
537 template <>
538 struct collection_dheader_traits<true> {
```

```
539     static bool dheader_required()
540     {
541         return is_dheader_required_in_enum_collections();
542     }
543 
544     static unsigned int dheader_size()
545     {
546         return dheader_required() ? RTI_XCDR_DHEADER_SIZE : 0;
547     }
548 };
549 
550 } // namespace detail
551 
552 // Forward declaration
553 template <typename T, unsigned int N>
554 class FinalArrayOffset;
555 
562 class OffsetBase {
563 public:
564 
582     bool is_null() const
583     {
584         return sample_ == NULL;
585     }
586 
595     bool is_cpp_compatible() const
596     {
597         // Derived classes may indicate whether
598         // a type is cpp-compatible
599         return false;
600     }
601 
607     unsigned char * get_buffer()
608     {
609         if (is_null()) {
610             return NULL;
611         }
612         return stream_.current_position();
613     }
614 
627     const unsigned char * get_buffer() const
628     {
629         if (is_null()) {
630             return NULL;
631         }
632         return stream_.current_position();
633     }
634 
641     offset_t get_buffer_size() const
642     {
643         RTI_FLAT_OFFSET_CHECK_NOT_NULL(return 0);
644         return stream_.total_size();
645     }
646 
652     friend bool operator<(
653             const OffsetBase & s1,
654             const OffsetBase & s2) {
655         return s1.get_buffer() < s2.get_buffer();
656     }
657 
663     friend bool operator > (
664             const OffsetBase & s1,
665             const OffsetBase & s2) {
666         return s1.get_buffer() > s2.get_buffer();
667     }
668 
674     friend bool operator <= (
675             const OffsetBase & s1,
676             const OffsetBase & s2) {
677         return s1.get_buffer() <= s2.get_buffer();
678     }
679 
685     friend bool operator >= (
686             const OffsetBase & s1,
687             const OffsetBase & s2) {
688         return s1.get_buffer() >= s2.get_buffer();
689     }
690 
696     friend bool operator == (
697             const OffsetBase & s1,
698             const OffsetBase & s2) {
```

```
699         return s1.get_buffer() == s2.get_buffer();
700     }
701
707     friend bool operator != (
708             const OffsetBase & s1,
709             const OffsetBase & s2) {
710         return !(s1 == s2);
711     }
712
713 protected:
714
715     OffsetBase() :
716             sample_(NULL),
717             absolute_offset_(0)
718     {
719     }
720
721     OffsetBase(
722             SampleBase *sample,
723             offset_t absolute_offset,
724             offset_t serialized_size)
725         : sample_(sample),
726           absolute_offset_(absolute_offset)
727     {
728         // In modern C++ this throws PreconditionNotMetError; in traditional
729         // C++, this ends the constructor leaving the object in a state such
730         // that is_null() is true.
731         RTI_FLAT_CHECK_PRECONDITION(sample != NULL, return);
732
733         sample_->initialize_stream(
734                 stream_.c_stream(),
735                 absolute_offset_,
736                 serialized_size);
737     }
738
739     // Gets the value of a primitive member at the specified relative Offset
740     template <typename U>
741     U deserialize(offset_t member_offset) const
742     {
743         RTI_FLAT_OFFSET_CHECK_NOT_NULL(return U());
744         RTI_FLAT_ASSERT(stream_.check_size(member_offset + static_cast<offset_t>(sizeof(U))), return U());
745
746         rti::xcdr::Stream::Memento stream_memento(stream_);
747         stream_.skip_fast(member_offset);
748         return stream_.deserialize_fast<U>();
749     }
750
751     // Sets the value of a primitive member at the specified relative Offset
752     template <typename U>
753     bool serialize(offset_t member_offset, U value)
754     {
755         RTI_FLAT_OFFSET_CHECK_NOT_NULL(return false);
756         RTI_FLAT_ASSERT(stream_.check_size(member_offset + static_cast<offset_t>(sizeof(U))), return false);
757
758         rti::xcdr::Stream::Memento stream_memento(stream_);
759
760         stream_.skip_fast(member_offset);
761         stream_.serialize_fast<U>(value);
762         return true;
763     }
764
765     // Obtains a member at the specified relative offset
766     //
767     // U is the type of the member. U must be a subclass of Offset<U>.
768     template <typename U>
769     U get_member(offset_t relative_member_offset)
770     {
771         RTI_FLAT_OFFSET_CHECK_NOT_NULL(return U());
772
773         return U(
774             sample_,
775             // the absolute location of the member:
776             absolute_offset_ + relative_member_offset);
777     }
778
779     template <typename U>
780     U get_member(offset_t relative_member_offset) const
781     {
782         RTI_FLAT_OFFSET_CHECK_NOT_NULL(return U());
783
784         return U(
```

```
785             sample_,
786             // the absolute location of the member:
787             absolute_offset_ + relative_member_offset);
788     }
789
790 protected:
791     SampleBase *sample_; // the actual CDR buffer is in the sample
792
793     offset_t absolute_offset_; // the position of this Offset in the buffer
794     mutable rti::xcdr::Stream stream_; // a substream that shares the same buffer
795 };
796
804 template <typename T>
805 class FinalOffset : public OffsetBase {
806 public:
807     typedef fixed_size_type_tag_t offset_kind;
808
809     static offset_t serialized_size_w_padding()
810     {
811         RTI_FLAT_ASSERT(T::serialized_size(0) > 0, return 0);
812         RTI_FLAT_ASSERT(T::required_alignment > 0, return 0);
813
814         const offset_t element_size = T::serialized_size(0);
815         if (element_size % T::required_alignment != 0) {
816            return element_size + T::required_alignment
817                    - element_size % T::required_alignment;
818         } else {
819             return element_size;
820         }
821     }
822
823     bool is_cpp_compatible() const // override
824     {
825         RTI_FLAT_OFFSET_CHECK_NOT_NULL(return false);
826         return !stream_.needs_byte_swap()
827                 && rti::xcdr::has_cpp_friendly_cdr_layout<
828                         typename rti::flat::flat_type_traits<T>::flat_type>();
829     }
830
831 protected:
832     FinalOffset()
833     {
834     }
835
836     // Creates an Offset within the sample's CDR  buffer at the specified
837     // absolute offset
838     //
839     // This overload requires T::serialized_size(offset_t)
840     //
841     FinalOffset(SampleBase *sample, offset_t absolute_offset)
842         : OffsetBase(
843                 sample,
844                 absolute_offset,
845                 T::serialized_size(absolute_offset))
846     {
847     }
848
849     // Gets the value of a primitive member at the specified relative offset
850     // relative_member_offsets - the 4 possible offsets depending on the this
851     // type's initial alignment
852     template <typename U>
853     U deserialize(const offset_t *relative_member_offsets) const
854     {
855         RTI_FLAT_ASSERT(relative_member_offsets != NULL, return U());
856
857         // In a FinalOffset the stream is guaranteed to have enough space to
858         // contain all its members. There are two possible cases:
859         //
860         // - If the FinalOffset is a member of a mutable type, the member getter
861         // will ensure that the stream has enough space for the whole final
862         // type.
863         //
864         // - If the final offset is a top-level type, the type plugin ensures
865         // that the serialized buffer is large enough to contain the type.
866
867         return OffsetBase::deserialize<U>(
868                 // pick the right offset based on the initial alignment
869                 get_value_for_alignment(relative_member_offsets));
870     }
871
872     // Sets the value of a primitive member at the specified relative offset
```

```
873     template <typename U>
874     bool serialize(const offset_t *relative_member_offsets, U value)
875     {
876         RTI_FLAT_ASSERT(relative_member_offsets != NULL, return false);
877 
878         return OffsetBase::serialize<U>(
879                 // pick the right offset based on the initial alignment
880                 get_value_for_alignment(relative_member_offsets), value);
881     }
882 
883 
884     // Obtains a member at the specified relative offset
885     //
886     // U is the type of the member. U must be a subclass of Offset<U>.
887     template <typename U>
888     U get_member(const offset_t *relative_member_offsets)
889     {
890         RTI_FLAT_ASSERT(relative_member_offsets != NULL, return U());
891 
892         return OffsetBase::get_member<U>(
893                 get_value_for_alignment(relative_member_offsets));
894     }
895 
896     template <typename U>
897     U get_member(const offset_t *relative_member_offsets) const
898     {
899         RTI_FLAT_ASSERT(relative_member_offsets != NULL, return U());
900 
901         return OffsetBase::get_member<U>(
902                 get_value_for_alignment(relative_member_offsets));
903     }
904 
905     // Returns an FinalArrayOffset located within this Offset
906     template <typename U, unsigned int N>
907     FinalArrayOffset<U, N> get_array_member(
908         const offset_t *relative_member_offsets,
909         const offset_t *first_element_sizes,
910         offset_t element_size)
911     {
912         typedef FinalArrayOffset<U, N> ArrayType;
913 
914         RTI_FLAT_ASSERT(relative_member_offsets != NULL, return ArrayType());
915         RTI_FLAT_ASSERT(first_element_sizes != NULL, return ArrayType());
916 
917         RTI_FLAT_OFFSET_CHECK_NOT_NULL(return ArrayType());
918 
919         return ArrayType(
920             sample_,
921             // the absolute location of the member:
922             absolute_offset_ + get_value_for_alignment(relative_member_offsets),
923             // the size of the first element, which can differ from the rest
924             get_value_for_alignment(first_element_sizes),
925             // the size of every other element
926             element_size);
927     }
928 
929     template <typename U, unsigned int N>
930     FinalArrayOffset<U, N> get_array_member(
931         const offset_t *relative_member_offsets,
932         const offset_t *first_element_sizes,
933         offset_t element_size) const
934     {
935         typedef FinalArrayOffset<U, N> ArrayType;
936 
937         RTI_FLAT_ASSERT(relative_member_offsets != NULL, return ArrayType());
938         RTI_FLAT_ASSERT(first_element_sizes != NULL, return ArrayType());
939 
940         RTI_FLAT_OFFSET_CHECK_NOT_NULL(return ArrayType());
941         return ArrayType(
942             sample_,
943             // the absolute location of the member:
944             absolute_offset_ + get_value_for_alignment(relative_member_offsets),
945             // the size of the first element, which can differ from the rest
946             get_value_for_alignment(first_element_sizes),
947             // the size of every other element
948             element_size);
949     }
950 
951 private:
952     // A member's relative offset or an array's first element size
953     // may be different depending on the alignment of the position in the CDR
```

```
954     // buffer of the type that contains them (this). These values are passed
955     // to get_member or get_array_member as an array of 4 possible values. This
956     // function picks the right one based on the alingmenet of absolute_offset_
957     offset_t get_value_for_alignment(const offset_t *values_per_alignment) const
958     {
959         return values_per_alignment
960                 [absolute_offset_ % RTI_XCDR_MAX_XCDR2_ALIGNMENT];
961     }
962 };
963
964 struct MutableOffsetHelper {
965     // Obtains the size of a mutable struct located at the absolute_offset of
966     // a FlatData sample
967     static offset_t calculate_serialized_size(
968             rti::flat::SampleBase *sample,
969             offset_t absolute_offset,
970             offset_t)
971     {
972         RTI_FLAT_ASSERT(sample != NULL, return 0);
973
974         return RTIXCdrFlatSample_getMutableSampleSize(
975                 sample->get_buffer(),
976                 absolute_offset);
977     }
978 };
979
987 class MutableOffset : public OffsetBase {
988 public:
989     typedef variable_size_type_tag_t offset_kind;
990     typedef MutableOffsetHelper Helper;
991
992 protected:
993     MutableOffset()
994     {
995     }
996
997     MutableOffset(
998             SampleBase *sample,
999             offset_t absolute_offset,
1000             offset_t serialized_size)
1001         : OffsetBase(
1002                 sample,
1003                 absolute_offset,
1004                 serialized_size)
1005     {
1006     }
1007
1008     template <typename U>
1009     U deserialize(member_id_t member_id, U default_val = U()) const
1010     {
1011         RTI_FLAT_OFFSET_CHECK_NOT_NULL(return U());
1012
1013         rti::xcdr::Stream::Memento stream_memento(stream_);
1014
1015         offset_t member_size = 0;
1016         if (!RTIXCdrStream_findV2MutableSampleMember(
1017                 &stream_.c_stream(),
1018                 member_id,
1019                 &member_size)) {
1020             return default_val;
1021         }
1022
1023         return stream_.deserialize_fast<U>();
1024     }
1025
1026     // Sets the value of a primitive member at the specified relative offset
1027     template <typename U>
1028     bool serialize(member_id_t member_id, U value)
1029     {
1030         RTI_FLAT_OFFSET_CHECK_NOT_NULL(return false);
1031
1032         rti::xcdr::Stream::Memento stream_memento(stream_);
1033
1034         offset_t member_size = 0;
1035         if (!RTIXCdrStream_findV2MutableSampleMember(
1036                 &stream_.c_stream(),
1037                 member_id,
1038                 &member_size)) {
1039             return false;
1040         }
1041
```

```
1042         stream_.serialize_fast<U>(value);
1043         return true;
1044     }
1045
1046     template <typename U>
1047     U get_member(member_id_t member_id) const
1048     {
1049         RTI_FLAT_OFFSET_CHECK_NOT_NULL(return U());
1050
1051         rti::xcdr::Stream::Memento stream_memento(stream_);
1052
1053         offset_t member_size = 0;
1054         if (!RTIXCdrStream_findV2MutableSampleMember(
1055                 &stream_.c_stream(),
1056                 member_id,
1057                 &member_size)) {
1058             return U();
1059         }
1060
1061         return get_member_impl<U>(
1062                 // absolute offset to the found member
1063                 detail::ptrdiff(stream_.current_position(), sample_->get_buffer()),
1064                 member_size,
1065                 typename U::offset_kind());
1066     }
1067
1068 private:
1069     template <typename U>
1070     U get_member_impl(
1071         offset_t absolute_member_offset,
1072         offset_t member_size,
1073         variable_size_type_tag_t) const
1074     {
1075         RTI_FLAT_ASSERT(absolute_member_offset > 0, return U());
1076
1077         // VariableOffsets receive a member_size
1078         return U(sample_, absolute_member_offset, member_size);
1079     }
1080
1081     template <typename U>
1082     U get_member_impl(
1083         offset_t absolute_member_offset,
1084         offset_t member_size,
1085         fixed_size_type_tag_t) const
1086     {
1087         (void) member_size; // suppress unused-param warning in release mode
1088         RTI_FLAT_ASSERT(absolute_member_offset > 0, return U());
1089
1090         // FixedOffsets do not receive a member_size (it's already known)
1091         return U(sample_, absolute_member_offset);
1092     }
1093 };
1094
1103 template <typename T>
1104 struct PrimitiveConstOffset : public OffsetBase {
1105 public:
1106     typedef fixed_size_type_tag_t offset_kind;
1107
1108     PrimitiveConstOffset()
1109     {
1110     }
1111
1112     PrimitiveConstOffset(
1113             SampleBase *sample,
1114             offset_t absolute_offset)
1115         : OffsetBase(
1116                 sample,
1117                 absolute_offset,
1118                 serialized_size(0))
1119     {
1120     }
1121
1122     bool is_cpp_compatible() const
1123     {
1124         RTI_FLAT_OFFSET_CHECK_NOT_NULL(return false);
1125         return sizeof(T) == 1 || !stream_.needs_byte_swap();
1126     }
1127
1133     T get() const
1134     {
1135         RTI_FLAT_OFFSET_CHECK_NOT_NULL(return T());
```

```
1136         RTI_FLAT_ASSERT(stream_.check_size(sizeof(T)), return T());
1137
1138         return stream_.deserialize_fast<T>();
1139     }
1140
1141     static offset_t serialized_size(offset_t)
1142     {
1143         return sizeof(T);
1144     }
1145
1146     static offset_t serialized_size_w_padding()
1147     {
1148         return serialized_size(0);
1149     }
1150
1151 };
1152
1169 template <typename T>
1170 struct PrimitiveOffset : public PrimitiveConstOffset<T> {
1171 public:
1172     PrimitiveOffset()
1173     {
1174     }
1175
1176     PrimitiveOffset(
1177             SampleBase *sample,
1178             offset_t absolute_offset)
1179         : PrimitiveConstOffset<T>(
1180                 sample,
1181                 absolute_offset)
1182     {
1183     }
1184
1190     bool set(T value)
1191     {
1192         RTI_FLAT_OFFSET_CHECK_NOT_NULL(return false);
1193         RTI_FLAT_ASSERT(this->stream_.check_size(sizeof(T)), return false);
1194
1195         this->stream_.template serialize_fast<T>(value);
1196         return true;
1197     }
1198 };
1199
1200 } }
1201
1202 #endif // RTI_DDS_FLAT_FLATOFFSETS_HPP_
1203
```

10.17 rtiflat.hpp

```
1 /*
2 (c) Copyright, Real-Time Innovations, 2018.
3 All rights reserved.
4
5 No duplications, whole or partial, manual or electronic, may be made
6 without express written permission.  Any such copies, or
7 revisions thereof, must display this notice unaltered.
8 This code contains trade secrets of Real-Time Innovations, Inc.
9 */
10
11 // Note: this file shouldn't be included directly. Include the header for the
12 // corresponding C++ API:
13 //
14 //  - Modern C++ API: "rti/topic/flat/FlatData.hpp"
15 //  - Traditional C++ API: "ndds/ndds_flat_data_cpp.h"
16
17 #ifndef RTI_DDS_FLAT_RTIFLAT_HPP_
18 #define RTI_DDS_FLAT_RTIFLAT_HPP_
19
154 #ifdef DOXYGEN_DOCUMENTATION_ONLY
155 namespace rti {
156
160     namespace flat {
161     }
162 }
163 #endif
164
```

```
165 #include "rti/flat/ExceptionHelper.hpp"
166 #include "rti/flat/FlatSample.hpp"
167 #include "rti/flat/FlatTypeTraits.hpp"
168 #include "rti/flat/SequenceBuilders.hpp"
169
170 #endif // RTI_DDS_FLAT_RTIFLAT_HPP_
171
```

10.18 SequenceBuilders.hpp

```
1 /*
2 (c) Copyright, Real-Time Innovations, 2018-2024.
3 All rights reserved.
4
5 No duplications, whole or partial, manual or electronic, may be made
6 without express written permission.  Any such copies, or
7 revisions thereof, must display this notice unaltered.
8 This code contains trade secrets of Real-Time Innovations, Inc.
9 */
10
11 #ifndef RTI_DDS_FLAT_SEQUENCEBUILDERS_HPP_
12 #define RTI_DDS_FLAT_SEQUENCEBUILDERS_HPP_
13
14 #include "rti/flat/Builder.hpp"
15
16 //
17 // Builders for sequences, strings, and arrays.
18 //
19 // Type hierarchy (- indicates an abstract builder; * a concrete, usable builder)
20 //
21 // - AbstractListBuilder
22 //     * MutableArrayBuilder
23 //     - AbstractSequenceBuilder
24 //         * MutableSequenceBuilder
25 //         * FinalSequenceBuilder
26 //         * PrimitiveSequenceBuilder
27 //             * StringBuilder
28 //
29 // Notes:
30 //   - there's no "FinalArrayBuilder" because that's a fixed-size type and
31 //     doesn't need a builder (it's constructed via an "add" operation, not a "build"
32 //     operation).
33 //   - MutableSequenceBuilder "builds" its elements
34 //   - FinalSequenceBuilder "adds" its elements
35 //
36
37 namespace rti { namespace flat {
38
39 namespace detail {
40
41 template <bool IsForEnum>
42 struct DHeaderGeneratorImpl {
43     static bool reserve_dheader(rti::xcdr::Stream& stream)
44     {
45         if (collection_dheader_traits<IsForEnum>::dheader_required()) {
46             return stream.serialize_dheader() != NULL;
47         } else {
48             return true;
49         }
50     }
51
52     static bool finish_dheader(
53             rti::xcdr::Stream& stream,
54             unsigned char *dheader_position)
55     {
56         if (collection_dheader_traits<IsForEnum>::dheader_required()) {
57             return stream.finish_dheader((char *) dheader_position);
58         } else {
59             return true;
60         }
61     }
62
63     static bool move_after_dheader(
64             rti::xcdr::Stream& stream,
65             unsigned char *begin_position)
66     {
67         stream.current_position(begin_position);
```

```
68
69          if (collection_dheader_traits<IsForEnum>::dheader_required()) {
70              return stream.skip(RTI_XCDR_DHEADER_SIZE);
71          } else {
72              return true;
73          }
74      }
75  };
76
77  struct NoDHeaderGenerator {
78      static bool reserve_dheader(rti::xcdr::Stream&)
79      {
80          return true;
81      }
82
83      static bool finish_dheader(rti::xcdr::Stream&, unsigned char *)
84      {
85          return true;
86      }
87
88      static bool move_after_dheader(
89              rti::xcdr::Stream& stream,
90              unsigned char *begin_position)
91      {
92          stream.current_position(begin_position);
93          return true;
94      }
95  };
96
97  typedef DHeaderGeneratorImpl<false> DHeaderGenerator;
98  typedef DHeaderGeneratorImpl<true> DHeaderGeneratorForEnum;
99
100 // Maybe specialized for enums to select DHeaderGenerator
101 template <typename T, typename Enable = void>
102 struct primitive_sequence_dheader_gen {
103     typedef NoDHeaderGenerator type;
104 };
105
106 } // namespace detail
107
108 /*i
109     * Extends and specializes AbstractBuilder to add or override the
110     * functionality required to build a collection of consecutive elements
111     * (sequence or array):
112     *
113     * - Add or build elements
114     * - Keep track of the element count
115     * - Handle element alignment
116     *
117     */
122 template <typename DHeaderGen>
123 class AbstractListBuilder : public AbstractBuilder {
124 protected:
125     AbstractListBuilder() : element_count_(0)
126     { }
127
128     AbstractListBuilder(
129             nested_tag_t,
130             AbstractBuilder& parent,
131             unsigned int alignment)
132             : AbstractBuilder(nested_tag_t(), parent, alignment),
133                 element_count_(0)
134     {
135         DHeaderGen::reserve_dheader(stream());
136     }
137
138     template <typename ElementBuilder>
139     ElementBuilder build_next()
140     {
141         rti::xcdr::Stream::Memento stream_memento(stream());
142         return build_element<ElementBuilder>(stream_memento);
143     }
144
145     template <typename ElementOffset>
146     ElementOffset add_next()
147     {
148         RTI_FLAT_BUILDER_CHECK_VALID(return ElementOffset());
149
150         // save state in case of error
151         rti::xcdr::Stream::Memento stream_memento(stream());
152
```

```
153         if (!stream().align(ElementOffset::required_alignment)) {
154             RTI_FLAT_BUILDER_OUT_OF_RESOURCES_ERROR(return ElementOffset());
155         }
156 
157         ElementOffset element = add_element<ElementOffset>();
158         if (element.is_null()) {
159             return ElementOffset();
160         }
161 
162         // Success: clear memento, increment element count
163         stream_memento.discard();
164         element_count_++;
165 
166         return element;
167     }
168 
169     template <typename ElementOffset>
170     void add_n(unsigned int count)
171     {
172         RTI_FLAT_BUILDER_CHECK_VALID(return);
173 
174         if (count == 0) {
175             return;
176         }
177 
178         if (!stream().align(ElementOffset::required_alignment)) {
179             RTI_FLAT_BUILDER_OUT_OF_RESOURCES_ERROR(return);
180         }
181 
182         offset_t elements_size = ElementOffset::serialized_size(0)
183                 + ElementOffset::serialized_size_w_padding() * (count - 1);
184         if (!stream().skip(elements_size)) {
185             RTI_FLAT_BUILDER_OUT_OF_RESOURCES_ERROR(return);
186         }
187 
188         element_count_ += count;
189     }
190 
194     unsigned int element_count() const
195     {
196         return element_count_;
197     }
198 
199 #if defined(RTI_FLAT_DATA_CXX11_RVALUE_REFERENCES)
200     AbstractListBuilder(AbstractListBuilder&& other) = default;
201     AbstractListBuilder& operator=(AbstractListBuilder&& other)
202     {
203         if (this == &other) {
204             return *this;
205         }
206 
207         finish_untyped_impl();
208 
209         AbstractBuilder::operator=(static_cast<AbstractBuilder&&>(other));
210 
211         element_count_ = other.element_count_;
212         other.element_count_ = 0;
213 
214         return *this;
215     }
216   #else
217 public:
218     // Enables the safe-move-constructor idiom without C++11 move constructors
219     struct AbstractListBuilderMoveProxy : AbstractBuilderMoveProxy {
220         unsigned int element_count_;
221     };
222 
223     operator AbstractListBuilderMoveProxy () throw() // move-constructor idiom
224     {
225         AbstractListBuilderMoveProxy other;
226         move_to(other);
227         return other;
228     }
229 
230 protected:
231     void move_from(AbstractListBuilderMoveProxy& other)
232     {
233         AbstractBuilder::move_from(other);
234         element_count_ = other.element_count_;
235     }
236 
```

```
237     void move_to(AbstractListBuilderMoveProxy& other)
238     {
239         AbstractBuilder::move_to(other);
240         other.element_count_ = element_count_;
241     }
242 
243 #endif
244 
245 private:
246     friend class AbstractBuilder;
247 
248     virtual void finish_member() // override
249     {
250         RTI_FLAT_BUILDER_CHECK_VALID(return);
251 
252         // Increment element count after the nested element  builder finishes,
253         // instead of on build_next(). This makes it easier to roll back the
254         // state if the element building doesn't complete.
255         element_count_++;
256         AbstractBuilder::finish_member();
257     }
258 
259 protected:
260     bool finish_dheader()
261     {
262         return DHeaderGen::finish_dheader(stream(), begin_position());
263     }
264 
265     unsigned int element_count_;
266 };
267 
268 // --- Array: ----------------------------------------------------------------
269 //
270 // (Note: there is no FinalArrayBuilder, because a final array's size is fixed,
271 // and is handled directly returning its offset with add_*; on the other hand,
272 // sequences are never fixed-size.)
273 
304 template <typename ElementBuilder, unsigned int N>
305 class MutableArrayBuilder : public AbstractListBuilder<detail::DHeaderGenerator> {
306 public:
310     typedef MutableArrayOffset<typename ElementBuilder::Offset, N> Offset;
311     typedef AbstractListBuilder<detail::DHeaderGenerator> Base;
312 
313     MutableArrayBuilder()
314     { }
315 
316     // The destructor does the same as finish_impl() but doesn't need to return
317     // an Offset
318     ~MutableArrayBuilder()
319     {
320         finish_untyped_impl();
321     }
322 
323 #if defined(RTI_FLAT_DATA_CXX11_RVALUE_REFERENCES)
324     MutableArrayBuilder(MutableArrayBuilder&& other) = default;
325 
326     MutableArrayBuilder& operator=(MutableArrayBuilder&& other)
327     {
328         finish_untyped_impl();
329         Base::operator=(static_cast<Base&&>(other));
330 
331         return *this;
332     }
333 #endif
334 
335     void finish_untyped_impl()
336     {
337         if (!Base::is_valid() || !Base::is_nested()) {
338             return;
339         }
340 
341         Base::finish_dheader();
342         Base::finish_untyped_impl();
343     }
344 
345 private:
346     friend class AbstractBuilder; // to allow access to the constructor
347 
348     MutableArrayBuilder(
349             nested_tag_t,
350             AbstractBuilder& parent,
```

```
351             unsigned int alignment = 0)
352         : AbstractListBuilder(nested_tag_t(), parent, alignment)
353     {
354     }
355
356 public:
363     ElementBuilder build_next()
364     {
365         if (element_count() == N) {
366             RTI_FLAT_BUILDER_PRECONDITION_ERROR(
367                 "Array builder build_next: too many elements",
368                 return ElementBuilder());
369         }
370
371         return AbstractListBuilder::build_next<ElementBuilder>();
372     }
373
381     Offset finish()
382     {
383         RTI_FLAT_BUILDER_CHECK_CAN_FINISH(return Offset());
384
385         if (element_count() != N) {
386             RTI_FLAT_BUILDER_PRECONDITION_ERROR(
387                 "Cannot finish array builder: too few elements",
388                 return Offset());
389         }
390
391         if (!finish_dheader()) {
392             RTI_FLAT_BUILDER_OUT_OF_RESOURCES_ERROR(return Offset());
393         }
394
395         return finish_impl<Offset>();
396     }
397
398     RTI_FLAT_BUILDER_DEFINE_MOVE_OPERATIONS_IMPL(\
399             MutableArrayBuilder, AbstractListBuilder, AbstractListBuilderMoveProxy)
400
401 };
402
403 // --- Sequences: ------------------------------------------------------------
404
405 // Specializes AbstractListBuilder to handle the length header serialization
406 // common to all sequences (which arrays don't have)
414 template <typename DHeaderGen>
415 class AbstractSequenceBuilder : public AbstractListBuilder<DHeaderGen> {
416 public:
417     typedef AbstractListBuilder<DHeaderGen> Base;
418     using Base::element_count;
419     using Base::is_nested;
420     using Base::is_valid;
421     using Base::stream;
422     using typename Base::nested_tag_t;
423   #if !defined(RTI_FLAT_DATA_CXX11_RVALUE_REFERENCES)
424     using typename Base::AbstractListBuilderMoveProxy;
425   #endif
426
427 protected:
428
429     AbstractSequenceBuilder()
430     {
431     }
432
433     AbstractSequenceBuilder(
434             nested_tag_t,
435             AbstractBuilder& parent,
436             unsigned int alignment)
437             : Base(nested_tag_t(), parent, alignment)
438     {
439         // leave space for sequence length
440         if (!stream().check_size(sizeof(rti::xcdr::length_t))) {
441             RTI_FLAT_BUILDER_OUT_OF_RESOURCES_ERROR(this->invalidate());
442         } else {
443             // No need to align; the base constructor already aligned to 4 the
444             // begin_position
445             stream().template serialize_fast<rti::xcdr::length_t>(0);
446         }
447     }
448
449     // The destructor does the same as finish_impl() but doesn't need to return
450     // an Offset
451     ~AbstractSequenceBuilder()
```

```
452     {
453         finish_untyped_impl();
454     }
455 
456 #if defined(RTI_FLAT_DATA_CXX11_RVALUE_REFERENCES)
457     AbstractSequenceBuilder(AbstractSequenceBuilder&& other) = default;
458     AbstractSequenceBuilder& operator=(AbstractSequenceBuilder&& other)
459     {
460         finish_untyped_impl();
461         Base::operator=(static_cast<Base&&>(other));
462 
463         return *this;
464     }
465 #endif
466 
467     void finish_untyped_impl()
468     {
469         if (!Base::is_valid() || !Base::is_nested()) {
470             return;
471         }
472 
473         Base::finish_dheader();
474         finish_length();
475         Base::finish_untyped_impl();
476     }
477 
478     // Concrete sequences must call this method in their own finish() function
479     template <typename Offset>
480     Offset finish_impl()
481     {
482         RTI_FLAT_BUILDER_CHECK_VALID(return Offset());
483         RTI_FLAT_BUILDER_CHECK_CAN_FINISH(return Offset());
484 
485         Base::finish_dheader();
486         finish_length(); // serialize the length header
487         return Base::template finish_impl<Offset>();
488     }
489 
490 private:
491     // Serializes the sequence length when finishing the builder.
492     void finish_length()
493     {
494         rti::xcdr::Stream::Memento stream_memento(stream());
495         DHeaderGen::move_after_dheader(stream(), Base::begin_position());
496 
497         // Coverity found examples where element_count() is checked, but
498         // that's only done in Array builders, which are unrelated to this
499         // class, AbstractSequenceBuilder.
500         // coverity[CHECKED_RETURN : FALSE]
501         stream().template serialize_fast<rti::xcdr::length_t>(element_count());
502     }
503 };
504 
533 template <typename ElementBuilder>
534 class MutableSequenceBuilder
535         : public AbstractSequenceBuilder<detail::DHeaderGenerator> {
536 public:
540     typedef SequenceOffset<typename ElementBuilder::Offset> Offset;
541 
542     MutableSequenceBuilder()
543     {
544     }
545 
546 private:
547     typedef AbstractSequenceBuilder<detail::DHeaderGenerator> Base;
548 
549     friend class AbstractBuilder; // to allow access to the constructor
550 
551     MutableSequenceBuilder(
552             nested_tag_t,
553             AbstractBuilder& parent,
554             unsigned int alignment = RTI_XCDR_DHEADER_ALIGNMENT)
555         : Base(nested_tag_t(), parent, alignment)
556     {
557     }
558 
559 public:
566     ElementBuilder build_next()
567     {
568         return AbstractListBuilder::build_next<ElementBuilder>();
569     }
```

```
570
578     Offset finish()
579     {
580         return finish_impl<Offset>();
581     }
582
583     RTI_FLAT_BUILDER_DEFINE_MOVE_OPERATIONS_IMPL(
584             MutableSequenceBuilder,
585             Base,
586             AbstractListBuilderMoveProxy)
587 };
588
621 template <typename ElementOffset>
622 class FinalSequenceBuilder
623         : public AbstractSequenceBuilder<detail::DHeaderGenerator> {
624 public:
625     typedef SequenceOffset<ElementOffset> Offset;
626
627     FinalSequenceBuilder()
628     {
629     }
630
631 private:
632     friend class AbstractBuilder; // to allow access to the constructor
633
634     FinalSequenceBuilder(
635             nested_tag_t,
636             AbstractBuilder& parent,
637             unsigned int alignment = RTI_XCDR_SEQ_LENGTH_ALIGNMENT)
638         : AbstractSequenceBuilder(nested_tag_t(), parent, alignment)
639     {
640     }
641
642 public:
648     ElementOffset add_next()
649     {
650         return AbstractListBuilder::add_next<ElementOffset>();
651     }
652
661     FinalSequenceBuilder& add_n(unsigned int count)
662     {
663         AbstractListBuilder::add_n<ElementOffset>(count);
664         return *this;
665     }
666
673     Offset finish()
674     {
675         return finish_impl<Offset>();
676     }
677
678     RTI_FLAT_BUILDER_DEFINE_MOVE_OPERATIONS_IMPL(\
679             FinalSequenceBuilder, AbstractSequenceBuilder, AbstractListBuilderMoveProxy)
680 };
681
689 template <typename T>
690 class PrimitiveSequenceBuilder
691         : public AbstractSequenceBuilder<
692                 typename detail::primitive_sequence_dheader_gen<T>::type> {
693 public:
694     typedef PrimitiveSequenceOffset<T> Offset;
695     typedef AbstractSequenceBuilder<
696             typename detail::primitive_sequence_dheader_gen<T>::type>
697             Base;
698     using Base::is_valid;
699     using Base::stream;
700     using typename Base::nested_tag_t;
701 #if !defined(RTI_FLAT_DATA_CXX11_RVALUE_REFERENCES)
702     using typename Base::AbstractListBuilderMoveProxy;
703 #endif
704
705     PrimitiveSequenceBuilder()
706     {
707     }
708
709 protected:
710     friend class AbstractBuilder;  // to allow access to the constructor
711
712     PrimitiveSequenceBuilder(
713             nested_tag_t,
714             AbstractBuilder& parent,
715             unsigned int alignment = RTI_XCDR_SEQ_LENGTH_ALIGNMENT)
```

```
716             : Base(nested_tag_t(), parent, alignment)
717     {
718     }
719
720 public:
726     PrimitiveSequenceBuilder& add_next(T value)
727     {
728         RTI_FLAT_BUILDER_CHECK_VALID(return *this);
729
730         if (!stream().template serialize<T>(value)) {
731             RTI_FLAT_BUILDER_OUT_OF_RESOURCES_ERROR(return *this);
732         }
733
734         Base::element_count_++;
735         return *this;
736     }
737
746     PrimitiveSequenceBuilder& add_n(const T *array, unsigned int count)
747     {
748         RTI_FLAT_BUILDER_CHECK_VALID(return *this);
749         RTI_FLAT_ASSERT(array != NULL, return *this);
750
751         if (RTIXCdrUnsignedLong_MAX / static_cast<unsigned int>(sizeof(T))
752                 < count) {
753             RTI_FLAT_BUILDER_OUT_OF_RESOURCES_ERROR(return *this);
754         }
755
756         if (!stream().check_size(static_cast<unsigned int>(sizeof(T)) * count)) {
757             RTI_FLAT_BUILDER_OUT_OF_RESOURCES_ERROR(return *this);
758         }
759
760         if (stream().needs_byte_swap() && sizeof(T) > 1) {
761             for (unsigned int i = 0; i < count; i++) {
762                 stream().template serialize_fast<T>(array[i]);
763             }
764         } else {
765             stream().serialize_fast((void *) array, count * static_cast<unsigned int>(sizeof(T)));
766         }
767
768         Base::element_count_ += count;
769         return *this;
770     }
771
785     PrimitiveSequenceBuilder& add_n(unsigned int count, T value)
786     {
787         RTI_FLAT_BUILDER_CHECK_VALID(return *this);
788
789         if (RTIXCdrUnsignedLong_MAX / static_cast<unsigned int>(sizeof(T))
790                 < count) {
791             RTI_FLAT_BUILDER_OUT_OF_RESOURCES_ERROR(return *this);
792         }
793
794         if (!stream().check_size(static_cast<unsigned int>(sizeof(T)) * count)) {
795             RTI_FLAT_BUILDER_OUT_OF_RESOURCES_ERROR(return *this);
796         }
797
798         for (unsigned int i = 0; i < count; i++) {
799             stream().template serialize_fast<T>(value);
800         }
801         Base::element_count_ += count;
802         return *this;
803     }
804
825     PrimitiveSequenceBuilder& add_n(unsigned int count)
826     {
827         RTI_FLAT_BUILDER_CHECK_VALID(return *this);
828
829         if (RTIXCdrUnsignedLong_MAX / static_cast<unsigned int>(sizeof(T))
830                 < count) {
831             RTI_FLAT_BUILDER_OUT_OF_RESOURCES_ERROR(return *this);
832         }
833
834         if (!stream().skip(static_cast<unsigned int>(sizeof(T)) * count)) {
835             RTI_FLAT_BUILDER_OUT_OF_RESOURCES_ERROR(return *this);
836         }
837
838         Base::element_count_ += count;
839         return *this;
840     }
841
842
```

```
849     Offset finish()
850     {
851         this->finish_dheader();
852         return Base::template finish_impl<Offset>();
853     }
854
855 private:
856     RTI_FLAT_BUILDER_DEFINE_MOVE_OPERATIONS_IMPL(\
857             PrimitiveSequenceBuilder, Base, AbstractListBuilderMoveProxy)
858 };
859
875 class StringBuilder : public PrimitiveSequenceBuilder<char> {
876 public:
877     typedef StringOffset Offset;
878     typedef PrimitiveSequenceBuilder<char> Base;
879     using Base::is_valid;
880     using Base::stream;
881     using typename Base::nested_tag_t;
882
883     StringBuilder()
884     {
885     }
886
887 private:
888     friend class AbstractBuilder; // to allow access to the constructor
889
890     StringBuilder(
891             nested_tag_t,
892             AbstractBuilder& parent,
893             unsigned int alignment = RTI_XCDR_SEQ_LENGTH_ALIGNMENT)
894             : Base(nested_tag_t(), parent, alignment)
895     {
896     }
897
898 public:
906     StringBuilder& set_string(const char *value)
907     {
908         RTI_FLAT_BUILDER_CHECK_VALID(return *this);
909         RTI_FLAT_ASSERT(value != NULL, return *this);
910
911         // if set_string is called more than once we override the string
912         // that was set before
913         if (Base::element_count_ != 0) {
914             stream().current_position(begin_position());
915             stream().serialize_fast<rti::xcdr::length_t>(0);
916             Base::element_count_ = 0;
917         }
918
919         unsigned int length =
920                 static_cast<unsigned int>(RTIXCdrString_getLength(value)) + 1;
921         add_n(value, length);  // if this fail, error has been reported
922         return *this;
923     }
924
931     Offset finish()
932     {
933         RTI_FLAT_BUILDER_CHECK_VALID(return Offset());
934
935         if (element_count_ == 0) {
936             add_next('\0'); // build empty string if no string was built
937         }
938
939         Base::finish_dheader();
940         return finish_impl<Offset>();
941     }
942
943     RTI_FLAT_BUILDER_DEFINE_MOVE_OPERATIONS_IMPL(\
944             StringBuilder, Base, AbstractListBuilderMoveProxy)
945 };
946
947 // Wide strings are treated as a sequence of octets
948 typedef PrimitiveSequenceBuilder<unsigned char> WStringBuilder;
949
950 } }
951
952 #endif // RTI_DDS_FLAT_SEQUENCEBUILDERS_HPP_
```

10.19 SequenceIterator.hpp

```
1 /*
2 (c) Copyright, Real-Time Innovations, 2018.
3 All rights reserved.
4
5 No duplications, whole or partial, manual or electronic, may be made
6 without express written permission.  Any such copies, or
7 revisions thereof, must display this notice unaltered.
8 This code contains trade secrets of Real-Time Innovations, Inc.
9 */
10
11 #ifndef RTI_DDS_FLAT_SEQUENCEITERATOR_HPP_
12 #define RTI_DDS_FLAT_SEQUENCEITERATOR_HPP_
13
14 #include <iterator>
15
16 #include "rti/flat/Offset.hpp"
17
18 namespace rti { namespace flat {
19
43 template <typename E, typename OffsetKind>
44 class SequenceIterator {
45 public:
49     typedef std::forward_iterator_tag iterator_category;
50
54     typedef E value_type;
55
59     typedef value_type reference;
60
64     typedef value_type pointer;
65
69     typedef std::ptrdiff_t difference_type;
70
74     SequenceIterator() : sample_(NULL), current_offset_(0), max_offset_(0)
75     {
76     }
77
78     SequenceIterator(
79             rti::flat::SampleBase *sample,
80             offset_t initial_offset,
81             offset_t max_offset)
82         : sample_(sample),
83           current_offset_(initial_offset),
84           max_offset_(max_offset)
85     {
86         RTI_FLAT_ASSERT(sample != NULL, return);
87     }
88
92     bool is_null() const
93     {
94         return sample_ == NULL;
95     }
96
105     value_type operator*() const
106     {
107         return get_impl(OffsetKind());
108     }
109
113     value_type operator->() const
114     {
115         return get_impl(OffsetKind());
116     }
117
132     bool advance()
133     {
134         // Unlike operator++, advance() allows checking for errors when
135         // exceptions are disabled (traditional C++)
136         return advance_impl(OffsetKind());
137     }
138
147     SequenceIterator& operator++()
148     {
149         bool can_advance_iter = advance_impl(OffsetKind());
150         RTI_FLAT_CHECK_PRECONDITION(can_advance_iter, return *this);
151         return *this;
152     }
153
162     SequenceIterator operator++(int)
163     {
```

```
164         SequenceIterator tmp(*this);
165         ++(*this);
166         return tmp;
167     }
168
172     friend bool operator<(
173             const SequenceIterator & s1,
174             const SequenceIterator & s2)
175     {
176         return s1.get_position() < s2.get_position();
177     }
178
182     friend bool operator > (
183             const SequenceIterator & s1,
184             const SequenceIterator & s2)
185     {
186         return s1.get_position() > s2.get_position();
187     }
188
192     friend bool operator <= (
193             const SequenceIterator & s1,
194             const SequenceIterator & s2)
195     {
196         return s1.get_position() <= s2.get_position();
197     }
198
202     friend bool operator >= (
203             const SequenceIterator & s1,
204             const SequenceIterator & s2)
205     {
206         return s1.get_position() >= s2.get_position();
207     }
208
212     friend bool operator == (
213             const SequenceIterator & s1,
214             const SequenceIterator & s2)
215     {
216         return s1.get_position() == s2.get_position();
217     }
218
222     friend bool operator != (
223             const SequenceIterator & s1,
224             const SequenceIterator & s2)
225     {
226         return !(s1 == s2);
227     }
228
229     unsigned char * get_position() const
230     {
231         if (is_null()) {
232             return NULL;
233         } else {
234             return sample_->get_buffer() + current_offset_;
235         }
236     }
237
238 private:
239     E get_impl(variable_size_type_tag_t) const
240     {
241         if (is_null()) {
242             return E();
243         }
244
245         // The iterator points to the end of the previous element. If elements
246         // need padding, we have to add it here (required alignment is 4 because
247         // mutable types (variable_size_type_tag_t) are always aligned to 4)
248         offset_t offset = RTIXCdrAlignment_alignSizeUp(current_offset_, 4);
249
250         if (offset >= max_offset_) {
251             return E();
252         }
253
254         // Calculate the size of the current element; if E is a struct type,
255         // this will be quick (just deserialize the DHeader); if we're itearating
256         // over an array of sequences (E is a sequence), this will require
257         // skipping over the current sequence.
258         offset_t size = E::Helper::calculate_serialized_size(
259                 sample_,
260                 offset,
261                 max_offset_);
262         // size 0 indicates an error; a variable-size type E cannot have size
```

```
263         // zero because it will always have a header (DHeader for mutable structs,
264         // length for sequences)
265         if (size == 0) {
266             return E();
267         }
268 
269         return E(sample_, offset, size);
270     }
271 
272     E get_impl(fixed_size_type_tag_t) const
273     {
274         if (is_null()) {
275             return E();
276         }
277 
278         // The iterator points to the end of the previous element. If elements
279         // need padding, we have to add it here.
280         offset_t offset = RTIXCdrAlignment_alignSizeUp(
281                 current_offset_,
282                 E::required_alignment);
283         if (offset >= max_offset_) {
284             return E();
285         }
286 
287         return E(sample_, offset);
288     }
289 
290     bool advance_impl(variable_size_type_tag_t)
291     {
292         if (is_null()) {
293             return false;
294         }
295 
296         current_offset_ = RTIXCdrAlignment_alignSizeUp(current_offset_, 4);
297         offset_t size = E::Helper::calculate_serialized_size(
298                 sample_, // beginning of the buffer
299                 current_offset_, // absolute offset to the current element
300                 max_offset_); // maximum offset in the list
301         if (size == 0) {
302             return false;
303         }
304 
305         current_offset_ += size;
306 
307         if (current_offset_ > max_offset_) {
308             current_offset_ = max_offset_;
309             return false;
310         }
311 
312         return true;
313     }
314 
315     bool advance_impl(fixed_size_type_tag_t)
316     {
317         if (is_null()) {
318             return false;
319         }
320 
321         // The iterator points to the end of the previous element. If elements
322         // need padding, we have to add it here. We do that to keep the end()
323         // iterator consistent, becuase the last element may not need padding.
324         current_offset_ = RTIXCdrAlignment_alignSizeUp(
325                 current_offset_,
326                 E::required_alignment);
327         current_offset_ += E::serialized_size(0);
328 
329         if (current_offset_ > max_offset_) {
330             current_offset_ = max_offset_;
331             return false;
332         }
333 
334         return true;
335     }
336 
337     rti::flat::SampleBase *sample_;
338     offset_t current_offset_;
339     offset_t max_offset_;
340 };
341 
342 } }
343 
```

```
344 #endif // RTI_DDS_FLAT_SEQUENCEITERATOR_HPP_
345
```

10.20 SequenceOffsets.hpp

```
1 /*
2 (c) Copyright, Real-Time Innovations, 2018.
3 All rights reserved.
4
5 No duplications, whole or partial, manual or electronic, may be made
6 without express written permission.  Any such copies, or
7 revisions thereof, must display this notice unaltered.
8 This code contains trade secrets of Real-Time Innovations, Inc.
9 */
10
11 #ifndef RTI_DDS_FLAT_SEQUENCEOFFSETS_HPP_
12 #define RTI_DDS_FLAT_SEQUENCEOFFSETS_HPP_
13
14 #include "rti/flat/Offset.hpp"
15 #include "rti/flat/SequenceIterator.hpp"
16
17 namespace rti { namespace flat {
18
26 template <typename T>
27 class AbstractPrimitiveList : public OffsetBase {
28 public:
29     typedef T value_type;
30
31 protected:
32     AbstractPrimitiveList()
33     {
34     }
35
36     AbstractPrimitiveList(
37             rti::flat::SampleBase *sample,
38             offset_t offset,
39             offset_t sequence_size)
40         : OffsetBase(sample, offset, sequence_size)
41     {
42     }
43
44 public:
45
46     bool is_cpp_compatible() const
47     {
48         RTI_FLAT_OFFSET_CHECK_NOT_NULL(return false);
49         return sizeof(T) == 1 || !stream_.needs_byte_swap();
50     }
51
52
53     /*@private
54      * @brief Gets direct access to the elements
55      *
56      * Returns a pointer to the elements of this sequence.
57      *
58      * @note If the Sample that contains this PrimitiveSequenceOffset has an endianess
59      * different from the native one, the element bytes may be swapped. This can
60      * happen if this Sample was published by a DataWriter running on a platform
61      * with a different endianess and received by a DataReader running locally.
62      *
63      * This function is not for public use. See rti::flat::plain_cast() instead.
64      */
65     T * get_elements()
66     {
67         return reinterpret_cast<T*>(this->get_buffer());
68     }
69
70     /*@private
71      * @brief Gets direct access to the array elements
72      *
73      * (Const overload)
74      */
75     const T * get_elements() const
76     {
77         return reinterpret_cast<const T*>(this->get_buffer());
78     }
79
```

```
88      T get_element(unsigned int i) const
89      {
90          // OffsetBase::serialize only checks this as a debug assertion; in
91          // a sequence or array we need to do it in release mode too
92          if (!stream_.check_size((i + 1) * ((unsigned int) sizeof(T)))) {
93              return T();
94          }
95
96          return OffsetBase::template deserialize<T>(i * (unsigned int) sizeof(T));
97      }
98
111     bool set_element(unsigned int i, T value)
112     {
113         // OffsetBase::serialize only checks this as a debug assertion; in
114         // a sequence or array we need to do it in release mode too
115         if (!stream_.check_size((i + 1) * static_cast<unsigned int>(sizeof(T)))) {
116             return false;
117         }
118
119         return OffsetBase::serialize(i * static_cast<unsigned int>(sizeof(T)), value);
120     }
121 };
122
123 namespace detail {
124
125 template <typename T>
126 struct PrimitiveSequenceOffsetHelper;
127
128 template <typename T, typename Enable = void>
129 struct primitive_sequence_header_length {
130     static size_t value()
131     {
132         return sizeof(rti::xcdr::length_t);
133     }
134 };
135
136 } // namespace detail
137
145 template <typename T>
146 class PrimitiveSequenceOffset : public AbstractPrimitiveList<T> {
147 public:
148     typedef variable_size_type_tag_t offset_kind;
149     typedef detail::PrimitiveSequenceOffsetHelper<T> Helper;
150
151     PrimitiveSequenceOffset() : element_count_(0)
152     {
153     }
154
155     PrimitiveSequenceOffset(
156             rti::flat::SampleBase *sample,
157             offset_t offset,
158             offset_t serialized_size)
159         : AbstractPrimitiveList<T>(
160                 // passing NULL causes the base constructors to fail
161                 serialized_size >= static_cast<offset_t>(header_length()) ? sample : NULL,
162                 offset + static_cast<offset_t>(header_length()),
163                 serialized_size - static_cast<offset_t>(header_length())),
164         element_count_(0)
165     {
166 #ifdef RTI_FLAT_DATA_NO_EXCEPTIONS
167         if (!this->is_null()) { // without exceptions, we need to check if the base ctor failed
168 #endif
169         rti::xcdr::Stream::Memento stream_memento(this->stream_);
170         this->stream_.skip_back(sizeof(rti::xcdr::length_t));
171         element_count_ = this->stream_.template deserialize_fast<rti::xcdr::length_t>();
172 #ifdef RTI_FLAT_DATA_NO_EXCEPTIONS
173         }
174 #endif
175
176     }
177
181     unsigned int element_count() const
182     {
183         RTI_FLAT_OFFSET_CHECK_NOT_NULL(return 0);
184
185         return element_count_;
186     }
187
188 private:
189     static size_t header_length()
190     {
```

```
191         return detail::primitive_sequence_header_length<T>::value();
192     }
193
194     unsigned int element_count_;
195
196 };
197
198 namespace detail {
199
200 template <typename T>
201 struct PrimitiveSequenceOffsetHelper {
202     static offset_t calculate_serialized_size(
203             rti::flat::SampleBase *sample,
204             offset_t absolute_offset,
205             offset_t max_size)
206     {
207         RTI_FLAT_ASSERT(sample != NULL, return 0);
208         RTI_FLAT_ASSERT(max_size > sizeof(rti::xcdr::length_t), return 0);
209
210         PrimitiveSequenceOffset<T> tmp(sample, absolute_offset, max_size);
211
212         // elements + header
213         if ((RTIXCdrUnsignedLong_MAX
214                 - static_cast<unsigned int>(sizeof(rti::xcdr::length_t)))
215                 / static_cast<unsigned int>(sizeof(T))
216                 < tmp.element_count()) {
217             RTI_FLAT_OFFSET_PRECONDITION_ERROR(
218                     "Serialized size of a sequence of primitive elements "
219                     "exceeds maximum representable 32-bit unsigned integer.",
220                     return 0);
221         }
222         return (tmp.element_count() * static_cast<unsigned int>(sizeof(T)))
223                 + static_cast<unsigned int>(sizeof(rti::xcdr::length_t));
224     }
225 };
226
227 } // namespace detail
228
240 template <typename T, unsigned int N>
241 class PrimitiveArrayOffset : public AbstractPrimitiveList<T> {
242 public:
243     typedef fixed_size_type_tag_t offset_kind;
244
245     PrimitiveArrayOffset()
246     {
247     }
248
249     PrimitiveArrayOffset(rti::flat::SampleBase *sample, offset_t offset)
250         : AbstractPrimitiveList<T>(
251                 sample,
252                 offset + static_cast<offset_t>(header_length()),
253                 serialized_size(0))
254     {
255     }
256
257 public:
258
263     unsigned int element_count() const
264     {
265         return N;
266     }
267
268     static offset_t serialized_size(offset_t)
269     {
270         return sizeof(T) * N;
271     }
272
273 private:
274     static size_t header_length()
275     {
276         return detail::primitive_sequence_header_length<T>::value()
277                 - sizeof(rti::xcdr::length_t);
278     }
279 };
280
281 // Note: the modern C++ API defines a PrimitiveArrayOffset specializations for
282 // safe_enum and wchar_t
283
284 struct StringOffsetHelper;
285
291 class StringOffset : public PrimitiveSequenceOffset<char> {
```

```
292 public:
293     typedef variable_size_type_tag_t offset_kind;
294     typedef StringOffsetHelper Helper;
295 
296     StringOffset()
297     {
298     }
299 
300     StringOffset(
301             SampleBase *sample,
302             offset_t absolute_offset,
303             offset_t serialized_size)
304         : PrimitiveSequenceOffset<char>(
305                 sample,
306                 absolute_offset,
307                 serialized_size)
308     {
309     }
310 
316     char * get_string()
317     {
318         return reinterpret_cast<char*>(this->get_buffer());
319     }
320 
325     const char * get_string() const
326     {
327         return reinterpret_cast<const char*>(this->get_buffer());
328     }
329 
335     unsigned int element_count() const
336     {
337         RTI_FLAT_CHECK_PRECONDITION(
338                 PrimitiveSequenceOffset<char>::element_count() > 0,
339                 return 0);
340 
341         return PrimitiveSequenceOffset<char>::element_count() - 1;
342     }
343 };
344 
345 struct StringOffsetHelper {
346     static offset_t calculate_serialized_size(
347             rti::flat::SampleBase *sample,
348             offset_t absolute_offset,
349             offset_t max_size)
350     {
351         StringOffset tmp(sample, absolute_offset, max_size);
352         return tmp.element_count()
353             + 1 // null terminator
354             + (unsigned int) sizeof(rti::xcdr::length_t); // length header
355     }
356 };
357 
358 // Wide strings are treated as a sequence of octets
359 typedef PrimitiveSequenceOffset<unsigned char> WStringOffset;
360 
361 // The base class of SequenceOffset, and FinalAlignedArrayOffset
362 //
363 // This encapsulates the functionality that allows iterating through a list
364 // of elements (a sequence or an array). The list must be always aligned to 4
365 // (that excludes FinalArrayOffset)
366 //
373 template <typename ElementOffset>
374 class AbstractAlignedList : public OffsetBase {
375 public:
376 
380     typedef SequenceIterator<ElementOffset, typename ElementOffset::offset_kind>
381             iterator;
382 
383 protected:
384     AbstractAlignedList()
385     {
386     }
387 
388     AbstractAlignedList(
389             rti::flat::SampleBase *sample,
390             offset_t offset,
391             offset_t sequence_size)
392         : OffsetBase(
393                 sample,
394                 offset,
395                 sequence_size)
```

```
396     {
397     }
398
399 public:
400     bool is_cpp_compatible() const // override
401     {
402         RTI_FLAT_OFFSET_CHECK_NOT_NULL(return false);
403         return !stream_.needs_byte_swap()
404                 && rti::xcdr::has_cpp_friendly_cdr_layout_collection<
405                         typename rti::flat::flat_type_traits<ElementOffset>::flat_type>();
406     }
407
408     ElementOffset get_element(unsigned int i)
409     {
410         RTI_FLAT_OFFSET_CHECK_NOT_NULL(return ElementOffset());
411
412         return get_element_impl<ElementOffset>(
413                 i,
414                 typename ElementOffset::offset_kind());
415     }
416
429     iterator begin()
430     {
431         RTI_FLAT_OFFSET_CHECK_NOT_NULL(return iterator(NULL, 0, 0));
432
433         return iterator(
434                 sample_,
435                 absolute_offset_,
436                 absolute_offset_ + get_buffer_size());
437     }
438
442     iterator end()
443     {
444         RTI_FLAT_OFFSET_CHECK_NOT_NULL(return iterator(NULL, 0, 0));
445
446         return iterator(
447                 sample_,
448                 absolute_offset_ + get_buffer_size(),
449                 absolute_offset_ + get_buffer_size());
450     }
451
452 private:
453     // for variable size types we skip element by element
454     template <typename E>
455     E get_element_impl(unsigned int i, variable_size_type_tag_t)
456     {
457         iterator it = begin();
458         while (it != end() && i > 0) {
459             if (!it.advance()) {
460                 return E();
461             }
462             i--;
463         }
464
465         return *it;
466     }
467
468     // for fixed-size types we support random access
469     template <typename E>
470     E get_element_impl(unsigned int i, fixed_size_type_tag_t)
471     {
472         offset_t size = i * E::serialized_size_w_padding();
473
474         // Ensure that the stream has space for i + 1 elements
475         if (!stream_.check_size(size + E::serialized_size(0))) {
476             return E();
477         }
478
479         return E(this->sample_, this->absolute_offset_ + size);
480     }
481 };
482
483 template <typename ElementOffset>
484 struct SequenceOffsetHelper;
485
500 template <typename ElementOffset>
501 class SequenceOffset : public AbstractAlignedList<ElementOffset> {
502 public:
503     typedef variable_size_type_tag_t offset_kind;
504     typedef SequenceOffsetHelper<ElementOffset> Helper;
505
```

```
506     SequenceOffset() : element_count_(0)
507     {
508     }
509
510     SequenceOffset(
511             rti::flat::SampleBase *sample,
512             offset_t offset,
513             offset_t sequence_size)
514             : AbstractAlignedList<ElementOffset>(
515                     // Member size in bytes must be > 4, otherwise let base ctor
516                     // fail
517                     sequence_size >= header_length() ? sample : NULL,
518                     // The offset begins after any padding to align to 4;
519                     // padding may be needed for example for an array of
520                     // sequence of final types; the end position of the previous
521                     // sequence may not be aligned
522                     RTIXCdrAlignment_alignSizeUp(
523                             offset,
524                             RTI_XCDR_SEQ_LENGTH_ALIGNMENT)
525                             + header_length(),
526                     sequence_size - header_length()),
527               element_count_(0)
528     {
529 #ifdef RTI_FLAT_DATA_NO_EXCEPTIONS
530         if (!this->is_null()) { // without exceptions, we need to check if the base ctor failed
531 #endif
532         rti::xcdr::Stream::Memento stream_memento(this->stream_);
533         this->stream_.skip_back(sizeof(rti::xcdr::length_t));
534         element_count_ = this->stream_.template deserialize_fast<rti::xcdr::length_t>();
535 #ifdef RTI_FLAT_DATA_NO_EXCEPTIONS
536         }
537 #endif
538     }
539
546     ElementOffset get_element(unsigned int i)
547     {
548         RTI_FLAT_OFFSET_CHECK_NOT_NULL(return ElementOffset());
549
550         if (i >= element_count_) {
551             return ElementOffset();
552         }
553         return AbstractAlignedList<ElementOffset>::get_element(i);
554     }
555
559     unsigned int element_count() const
560     {
561         RTI_FLAT_OFFSET_CHECK_NOT_NULL(return 0);
562
563         return element_count_;
564     }
565
566 private:
567     static offset_t header_length()
568     {
569         return static_cast<offset_t>(
570                 detail::collection_dheader_traits<false>::dheader_size()
571                 + sizeof(rti::xcdr::length_t));
572     }
573
574
575     unsigned int element_count_;
576 };
577
578 template <typename E>
579 struct SequenceOffsetHelper {
580     // Calculates the serialized size of a Sequence by skipping each element
581     static offset_t calculate_serialized_size(
582             rti::flat::SampleBase *sample,
583             offset_t absolute_offset,
584             offset_t max_offset)
585     {
586         // Create a SequenceOffset beginning at sample + absolute_offset,
587         // with a size we don't know, but no greater than
588         // max_offset - absolute_offset
589         SequenceOffset<E> tmp(
590                 sample,
591                 absolute_offset,
592                 max_offset - absolute_offset);
593         unsigned int count = tmp.element_count();
594         typename SequenceOffset<E>::iterator it = tmp.begin();
595         for (unsigned int i = 0; i < count; i++) {
```

```
596             if (!it.advance()) {
597                 return 0; // error
598             }
599         }
600 
601         return detail::ptrdiff(it.get_position(), sample->get_buffer())
602                 - absolute_offset;
603     }
604 };
605 
620 template <typename ElementOffset, unsigned int N>
621 class MutableArrayOffset : public AbstractAlignedList<ElementOffset> {
622 public:
623     typedef variable_size_type_tag_t offset_kind;
624 
625     MutableArrayOffset()
626     {
627     }
628 
629     MutableArrayOffset(
630             rti::flat::SampleBase *sample,
631             offset_t offset,
632             offset_t sequence_size)
633             : AbstractAlignedList<ElementOffset>(
634                     sequence_size >= header_length() ? sample : NULL,
635                     offset + header_length(),
636                     sequence_size - header_length())
637     {
638     }
639 
646     ElementOffset get_element(unsigned int i)
647     {
648         if (i >= N) {
649             return ElementOffset();
650         }
651 
652         return AbstractAlignedList<ElementOffset>::get_element(i);
653     }
654 
655 private:
656     static offset_t header_length()
657     {
658         return static_cast<offset_t>(
659                 detail::collection_dheader_traits<false>::dheader_size());
660     }
661 };
662 
682 template <typename ElementOffset, unsigned int N>
683 class FinalArrayOffset : public OffsetBase {
684 public:
685     typedef fixed_size_type_tag_t offset_kind;
686 
687     FinalArrayOffset()
688     {
689     }
690 
691     // Constructor used when this array is part of a fixed-size type, and
692     // therefore it's initial alignment is not known ahead of time
693     FinalArrayOffset(
694         rti::flat::SampleBase *sample,
695         offset_t absolute_offset,
696         offset_t first_element_size,
697         offset_t element_size)
698         : OffsetBase(
699                 sample,
700                 absolute_offset,
701                 first_element_size + element_size * (N - 1)),
702           first_element_size_(first_element_size),
703           element_size_(element_size)
704     {
705     }
706 
713     ElementOffset get_element(unsigned int i)
714     {
715         RTI_FLAT_OFFSET_CHECK_NOT_NULL(return ElementOffset());
716 
717         if (i >= N) {
718             return ElementOffset(); // null offset
719         }
720 
721         offset_t element_offset = this->absolute_offset_;
```

```
722         if (i > 0) {
723             element_offset += this->first_element_size_
724                 + (i - 1) * this->element_size_;
725         }
726
727         return ElementOffset(this->sample_, element_offset);
728     }
729
730 private:
731     offset_t first_element_size_;
732     offset_t element_size_;
733 };
734
775 template <typename ElementOffset, unsigned int N>
776 class FinalAlignedArrayOffset : public AbstractAlignedList<ElementOffset> {
777 public:
778     typedef fixed_size_type_tag_t offset_kind;
779
780     FinalAlignedArrayOffset()
781     {
782     }
783
784     // Constructor used when this array is part of a mutable type, and
785     // therefore it's aligned to 4 always
786     FinalAlignedArrayOffset(
787             rti::flat::SampleBase *sample,
788             offset_t absolute_offset)
789             : AbstractAlignedList<ElementOffset>(
790                     sample,
791                     absolute_offset
792                             + detail::collection_dheader_traits<
793                                     false>::dheader_size(),
794                     serialized_size(0))
795     { }
796
797     // serialized_size() only works when this array is part of a mutable type
798     // and we know it's aligned to 4
799     static offset_t serialized_size(offset_t)
800     {
801         // [element1][pad][element2][pad]...[elementN]
802         return ElementOffset::serialized_size_w_padding() * (N - 1)
803                 + ElementOffset::serialized_size(0);
804     }
805
815     ElementOffset get_element(unsigned int i)
816     {
817         RTI_FLAT_OFFSET_CHECK_NOT_NULL(return ElementOffset());
818
819         if (i >= N) {
820             return ElementOffset();
821         }
822
823         return AbstractAlignedList<ElementOffset>::get_element(i);
824     }
825 };
826
827 namespace detail {
828
829 // Arrays of complex elements may need a DHEADER
830 template <typename T, unsigned int N>
831 struct is_fixed_type_w_dheader<FinalAlignedArrayOffset<T, N> > {
832     static bool value()
833     {
834         return is_dheader_required_in_non_primitive_collections();
835     }
836 };
837
838
839 } // namespace detail
840
841 }}  // namespace rti::flat
842
843 #endif // RTI_DDS_FLAT_SEQUENCEOFFSETS_HPP_
844
```

10.21 Interpreter.hpp

```
1 /* $Id$
```

```
2
3 (c) Copyright, Real-Time Innovations, 2018.
4 All rights reserved.
5
6 No duplications, whole or partial, manual or electronic, may be made
7 without express written permission.  Any such copies, or
8 revisions thereof, must display this notice unaltered.
9 This code contains trade secrets of Real-Time Innovations, Inc.
10
11 ============================================================================= */
12
13 #ifndef RTI_XCDR_INTERPRETER_HPP_
14 #define RTI_XCDR_INTERPRETER_HPP_
15
16 #include "xcdr/xcdr_interpreter.h"
17
18
19 namespace rti { namespace xcdr {
20
21 /*i
22  * Provides the type code of a type.
23  *
24  * This struct is to be specialized in generated code for each TopicType.
25  * Specializations must define a static member function:
26  * RTIXCdrTypeCode * get()
27  */
28 template <typename TopicType>
29 struct type_code;
30
31 namespace detail {
32
33 template <typename TopicType, bool IsCollection>
34 bool has_cpp_friendly_cdr_layout_impl()
35 {
36     RTIXCdrUnsignedLongLong size;
37     RTIXCdrAlignment alignment;
38     RTIXCdrXTypesComplianceMask xTypesComplianceMask;
39
40     xTypesComplianceMask = RTIXCdrInterpreter_getGlobalXtypeComplianceMask();
41
42     return RTIXCdrTypeCode_hasCFriendlyCdrLayout(
43             type_code<TopicType>::get(),
44             &size,
45             &alignment,
46             /*
47              * The number of element is not important; it's only important to
48              * know if there's one or more than one
49              */
50             IsCollection ? 2 : 1,
51             RTI_XCDR_TRUE, /* v2Encapsulation */
52             RTIXCdrXTypesComplianceMask_isBitSet(
53                     xTypesComplianceMask,
54                     RTI_XCDR_XTYPES_DHEADER_IN_NON_PRIMITIVE_COLLECTIONS_BIT),
55             RTIXCdrXTypesComplianceMask_isBitSet(
56                     xTypesComplianceMask,
57                     RTI_XCDR_XTYPES_ENUM_AS_PRIMITIVE_IN_COLLECTIONS_BIT));
58 }
59
60 }
61
62 template <typename TopicType>
63 bool has_cpp_friendly_cdr_layout()
64 {
65     static bool result =
66         detail::has_cpp_friendly_cdr_layout_impl<TopicType, false>();
67     return result;
68 }
69
70 template <typename TopicType>
71 bool has_cpp_friendly_cdr_layout_collection()
72 {
73     static bool result =
74         detail::has_cpp_friendly_cdr_layout_impl<TopicType, true>();
75     return result;
76 }
77
78 struct NoOpPropertyConfigurator {
79     static void configure(RTIXCdrInterpreterProgramsGenProperty&)
80     {
81         // In general (trad. and micro C++), no additional properties.
82         //
```

```
83          // dds_cpp.2.0 defines a custom Configurator
84          //
85      }
86  };
87
88  /*i
89   * @brief Programs singleton for the to/from cdr functions
90   *
91   * The combination of the ProgramSingleton template parameters builds a single
92   * instance for the given type, program kinds and options.
93   * This will typically be instantiated in IDL-generated code.
94   *
95   * @tparam TopicType The topic-type these programs operate with
96   * @tparam ProgramKind A mask of the different programs to be generated
97   * @tparam ResolveAlias Value for RTIXCdrInterpreterProgramsGenProperty::resolveAlias
98   *          when creating the programs
99   * @tparam InlineStruct Value for RTIXCdrInterpreterProgramsGenProperty::inlineStruct
100  *          when creating the programs
101  * @tparam OptimizeEnum Value for RTIXCdrInterpreterProgramsGenProperty::optimizeEnum
102  *          when creating the programs
103  * @tparam KeysOnly (default false) Whether to generate programs only for key fields
104  * @tparam PropertyConfigurator (default no-op) A functor that configures
105  *          additional RTIXCdrInterpreterProgramsGenProperty to generate the
106  *          programs.
107  *
108  * get_instance() creates the singleton the first time it's called. The singleton
109  * is automatically deleted by the destructor of a static local variable.
110  */
111 template <
112     typename TopicType,
113     int ProgramKind,
114     bool ResolveAlias,
115     bool InlineStruct,
116     bool OptimizeEnum,
117     bool KeysOnly = false,
118     typename PropertyConfigurator = NoOpPropertyConfigurator>
119 struct ProgramsSingleton {
120 private:
121     ProgramsSingleton()
122     {
123         RTIXCdrInterpreterProgramsGenProperty property =
124                 RTIXCdrInterpreterProgramsGenProperty_INITIALIZER;
125
126         configure_property(property);
127
128         programs_ = RTIXCdrInterpreterPrograms_new(
129                 type_code<TopicType>::get(),
130                 &property,
131                 ProgramKind);
132     }
133
134     ~ProgramsSingleton()
135     {
136         if (programs_ != NULL) {
137             RTIXCdrInterpreterPrograms_delete(programs_);
138         }
139     }
140
141     static void configure_property(
142             RTIXCdrInterpreterProgramsGenProperty& property)
143     {
144         property.generateWithAllFields =
145                 KeysOnly ? RTI_XCDR_FALSE : RTI_XCDR_TRUE;
146         property.resolveAlias = ResolveAlias ? RTI_XCDR_TRUE : RTI_XCDR_FALSE;
147         property.inlineStruct = InlineStruct ? RTI_XCDR_TRUE : RTI_XCDR_FALSE;
148         property.optimizeEnum = OptimizeEnum ? RTI_XCDR_TRUE : RTI_XCDR_FALSE;
149         PropertyConfigurator::configure(property);
150     }
151
152     RTIXCdrInterpreterPrograms *programs_;
153
154 public:
155     // Returns the singleton instance. It can be NULL if the program creation
156     // failed.
157     static RTIXCdrInterpreterPrograms * get_instance()
158     {
159         static ProgramsSingleton<
160                 TopicType,
161                 ProgramKind,
162                 ResolveAlias,
163                 InlineStruct,
```

```
164                 OptimizeEnum,
165                 KeysOnly,
166                 PropertyConfigurator> instance;
167         return instance.programs_;
168     }
169
170     // Creates the programs returned by get_instance() again for this singleton.
171     // This is required in testing when the global interpreter properties are
172     // modified from test to test.
173     static RTIXCdrInterpreterPrograms *regenerate_programs()
174     {
175         RTIXCdrInterpreterPrograms *programs = get_instance();
176         if (programs == NULL) {
177             return NULL;
178         }
179
180         RTIXCdrInterpreterProgramsGenProperty property =
181                 RTIXCdrInterpreterProgramsGenProperty_INITIALIZER;
182         configure_property(property);
183
184         // finalize and re-initialize the programs. Note that get_instance()
185         // will continue returning the same object, but internally it's been
186         // re-initialized.
187         RTIXCdrInterpreterPrograms_finalize(programs);
188         if (!RTIXCdrInterpreterPrograms_initialize(
189                     programs,
190                     type_code<TopicType>::get(),
191                     &property,
192                     ProgramKind)) {
193             return NULL;
194         }
195
196         return programs;
197     }
198
199 };
200
201 // Returns a ProgramsSingleton with the programs that the generated
202 // to/from cdr functions need
203 template <
204     typename TopicType,
205     bool ResolveAlias,
206     bool InlineStruct,
207     bool OptimizeEnum,
208     typename PropertyConfigurator>
209 RTIXCdrInterpreterPrograms * get_cdr_serialization_programs_w_property_configurator()
210 {
211     return ProgramsSingleton<
212             TopicType,
213             RTI_XCDR_SER_PROGRAM
214                 | RTI_XCDR_DESER_PROGRAM
215                 | RTI_XCDR_GET_SER_SIZE_PROGRAM
216                 | RTI_XCDR_GET_MAX_SER_SIZE_PROGRAM,
217             ResolveAlias,
218             InlineStruct,
219             OptimizeEnum,
220             false,
221             PropertyConfigurator>::get_instance();
222 }
223
224 template <
225     typename TopicType,
226     bool ResolveAlias,
227     bool InlineStruct,
228     bool OptimizeEnum>
229 RTIXCdrInterpreterPrograms * get_cdr_serialization_programs()
230 {
231     return ProgramsSingleton<
232             TopicType,
233             RTI_XCDR_SER_PROGRAM
234                 | RTI_XCDR_DESER_PROGRAM
235                 | RTI_XCDR_GET_SER_SIZE_PROGRAM
236                 | RTI_XCDR_GET_MAX_SER_SIZE_PROGRAM,
237             ResolveAlias,
238             InlineStruct,
239             OptimizeEnum,
240             false,
241             NoOpPropertyConfigurator>::get_instance();
242 }
243
244 /*i
```

```
245  * Provides the interpreter programs for a type. (Currently it only provides
246  * the initialize_sample program used to initialize FlatData samples)
247  *
248  * This struct is to be specialized in generated code for each TopicType.
249  * Specializations must define a static member function:
250  * RTIXCdrInterpreterPrograms * get(), which will call
251  * get_cdr_initialization_programs() with the options (ResolveAlias, etc.) used
252  * to generate code for this type.
253  */
254 template <typename SampleType>
255 struct type_programs;
256
257 } } // namespace rti::xcdr
258
259 #endif // RTI_XCDR_INTERPRETER_HPP_
```

10.22 Stream.hpp

```
1 /* $Id$
2
3 (c) Copyright, Real-Time Innovations, 2018.
4 All rights reserved.
5
6 No duplications, whole or partial, manual or electronic, may be made
7 without express written permission.  Any such copies, or
8 revisions thereof, must display this notice unaltered.
9 This code contains trade secrets of Real-Time Innovations, Inc.
10
11 ============================================================================= */
12
13 #ifndef RTI_XCDR_STREAM_HPP_
14 #define RTI_XCDR_STREAM_HPP_
15
16 #include "xcdr/xcdr_stream.h"
17
18 namespace rti { namespace xcdr {
19
20
21 #define RTI_XCDR_SPECIALIZE_PRIMITIVE_TYPE_TRAITS_BODY(\
22         TYPE,\
23         CDR_TYPE,\
24         SERIALIZE_MACRO)\
25     static void serialize_fast(RTIXCdrStream *stream, const TYPE* v)\
26     {\
27         RTIXCdrStream_ ## SERIALIZE_MACRO ## ByteFast(stream, (CDR_TYPE *) v);\
28     }\
29     static void deserialize_fast(RTIXCdrStream *stream, TYPE* v)\
30     {\
31         RTIXCdrStream_de ## SERIALIZE_MACRO ## ByteFast(stream, (CDR_TYPE *) v);\
32     }\
33     static bool serialize(RTIXCdrStream *stream, const TYPE* v)\
34     {\
35         return RTIXCdrStream_ ## SERIALIZE_MACRO ## Byte(stream, (CDR_TYPE *) v, RTI_XCDR_TRUE);\
36     }\
37     static bool deserialize(RTIXCdrStream *stream, TYPE* v)\
38     {\
39         return RTIXCdrStream_de ## SERIALIZE_MACRO ## Byte(stream, (CDR_TYPE *) v, RTI_XCDR_TRUE);\
40     }\
41
42 #define RTI_XCDR_SPECIALIZE_PRIMITIVE_TYPE_TRAITS(\
43         TYPE,\
44         CDR_TYPE,\
45         SERIALIZE_MACRO)\
46 template <>\
47 struct primitive_type_traits<TYPE> {\
48 RTI_XCDR_SPECIALIZE_PRIMITIVE_TYPE_TRAITS_BODY(TYPE, CDR_TYPE, SERIALIZE_MACRO)\
49 }
50
51 #define RTI_XCDR_SPECIALIZE_PRIMITIVE_TYPE_TRAITS_W_ALIGNMENT_PARAM(\
52         TYPE,\
53         CDR_TYPE,\
54         SERIALIZE_MACRO,\
55         ALIGNMENT)\
56 template <>\
57 struct primitive_type_traits<TYPE> {\
58     static void serialize_fast(RTIXCdrStream *stream, const TYPE* v)\
59     {\
```

```
60          RTIXCdrStream_ ## SERIALIZE_MACRO ## ByteFast(stream, (CDR_TYPE *) v);\
61      }\
62      static void deserialize_fast(RTIXCdrStream *stream, TYPE* v)\
63      {\
64          RTIXCdrStream_de ## SERIALIZE_MACRO ## ByteFast(stream, (CDR_TYPE *) v);\
65      }\
66      static bool serialize(RTIXCdrStream *stream, const TYPE* v)\
67      {\
68          return RTIXCdrStream_ ## SERIALIZE_MACRO ## Byte(stream, (CDR_TYPE *) v, RTI_XCDR_TRUE, ALIGNMENT);\
69      }\
70      static bool deserialize(RTIXCdrStream *stream, TYPE* v)\
71      {\
72          return RTIXCdrStream_de ## SERIALIZE_MACRO ## Byte(stream, (CDR_TYPE *) v, RTI_XCDR_TRUE,
    ALIGNMENT);\
73      }\
74  }
75
76  #define RTI_XCDR_SPECIALIZE_PRIMITIVE_TYPE_TRAITS_NO_ALIGN_PARAM(\
77          TYPE,\
78          CDR_TYPE,\
79          SERIALIZE_MACRO)\
80  template <>\
81  struct primitive_type_traits<TYPE> {\
82      static void serialize_fast(RTIXCdrStream *stream, const TYPE* v)\
83      {\
84          RTIXCdrStream_ ## SERIALIZE_MACRO ## ByteFast(stream, (CDR_TYPE *) v);\
85      }\
86      static void deserialize_fast(RTIXCdrStream *stream, TYPE* v)\
87      {\
88          RTIXCdrStream_de ## SERIALIZE_MACRO ## ByteFast(stream, (CDR_TYPE *) v);\
89      }\
90      static bool serialize(RTIXCdrStream *stream, const TYPE* v)\
91      {\
92          return RTIXCdrStream_ ## SERIALIZE_MACRO ## Byte(stream, (CDR_TYPE *) v);\
93      }\
94      static bool deserialize(RTIXCdrStream *stream, TYPE* v)\
95      {\
96          return RTIXCdrStream_de ## SERIALIZE_MACRO ## Byte(stream, (CDR_TYPE *) v);\
97      }\
98  }
99
100 // Base case handles T as an 4-byte integer (this is necessary for traditional
101 // C++ enums as well as C++11 enum classes)
102 template <typename T>
103 struct primitive_type_traits {
104     RTI_XCDR_SPECIALIZE_PRIMITIVE_TYPE_TRAITS_BODY(T, RTIXCdr4Byte, serialize4)
105 };
106
107 RTI_XCDR_SPECIALIZE_PRIMITIVE_TYPE_TRAITS_NO_ALIGN_PARAM(\
108         char, RTIXCdr1Byte, serialize1);
109
110 // The API can work with fixed-width integers (dds_cpp.2.0) or
111 // RTI-defined integers (dds_cpp.1.0). We need to make this distinction because
112 // on some platforms these types may not be exactly the same as far as template
113 // resolution. That is, compilers may or may not pick the template
114 // specialization for int32_t for an instantiation with int.
115 //
116 // If we define only one (e.g. int32_t) some compilers won't find a suitable
117 // definition for int. If we define both, some compilers will find duplicate
118 // definitions for int/int32_t.
119 #if !defined(RTI_AVOID_FIXED_WIDTH_INTEGERS)
120 RTI_XCDR_SPECIALIZE_PRIMITIVE_TYPE_TRAITS_NO_ALIGN_PARAM(\
121         uint8_t, RTIXCdr1Byte, serialize1);
122
123 RTI_XCDR_SPECIALIZE_PRIMITIVE_TYPE_TRAITS(\
124         int16_t, RTIXCdr2Byte, serialize2);
125 RTI_XCDR_SPECIALIZE_PRIMITIVE_TYPE_TRAITS(\
126         uint16_t, RTIXCdr2Byte, serialize2);
127
128 RTI_XCDR_SPECIALIZE_PRIMITIVE_TYPE_TRAITS(\
129         int32_t, RTIXCdr4Byte, serialize4);
130 RTI_XCDR_SPECIALIZE_PRIMITIVE_TYPE_TRAITS(\
131         uint32_t, RTIXCdr4Byte, serialize4);
132
133 typedef uint32_t length_t;
134 #else
135 RTI_XCDR_SPECIALIZE_PRIMITIVE_TYPE_TRAITS_NO_ALIGN_PARAM(\
136         unsigned char, RTIXCdr1Byte, serialize1);
137
138 RTI_XCDR_SPECIALIZE_PRIMITIVE_TYPE_TRAITS(\
139         short, RTIXCdr2Byte, serialize2);
```

```
140 RTI_XCDR_SPECIALIZE_PRIMITIVE_TYPE_TRAITS(\
141         unsigned short, RTIXCdr2Byte, serialize2);
142
143 // int and unsigned int, as well as enum are handled by the general case
144
145 typedef unsigned int length_t;
146 #endif
147
148 RTI_XCDR_SPECIALIZE_PRIMITIVE_TYPE_TRAITS(\
149         float, RTIXCdr4Byte, serialize4);
150
151 RTI_XCDR_SPECIALIZE_PRIMITIVE_TYPE_TRAITS_W_ALIGNMENT_PARAM(\
152         DDS_LongLong, RTIXCdr8Byte, serialize8, 4);
153
154 RTI_XCDR_SPECIALIZE_PRIMITIVE_TYPE_TRAITS_W_ALIGNMENT_PARAM(\
155         DDS_UnsignedLongLong, RTIXCdr8Byte, serialize8, 4);
156 RTI_XCDR_SPECIALIZE_PRIMITIVE_TYPE_TRAITS_W_ALIGNMENT_PARAM(\
157         double, RTIXCdr8Byte, serialize8, 4);
158
159 // RTI_XCDR_SPECIALIZE_PRIMITIVE_TYPE_TRAITS(
160 //         rti::core::LongDouble, RTI_CDR_LONG_DOUBLE_TYPE, RTIXCdr16Byte, serialize16Byte);
161
162 template <typename T>
163 void serialize_fast(RTIXCdrStream& stream, const T& value)
164 {
165     primitive_type_traits<T>::serialize_fast(&stream, &value);
166 }
167
168 template <typename T>
169 void deserialize_fast(RTIXCdrStream& stream, T& value)
170 {
171     primitive_type_traits<T>::deserialize_fast(&stream, &value);
172 }
173
174 class Stream {
175 public:
176     Stream()
177     {
178         RTIXCdrStream_init(&stream_);
179         RTIXCdrStream_set(&stream_, NULL, 0);
180
181         stream_._tmpRelativeBuffer = NULL;
182         stream_._needByteSwap = 0;
183         stream_._endian = stream_._nativeEndian;
184         stream_._encapsulationKind = 0;
185         stream_._encapsulationOptions = 0;
186     }
187
188     explicit Stream(const RTIXCdrStream& c_stream)
189     {
190         stream_ = c_stream; // shallow assign
191     }
192
193     // Creates a new stream that shares the same buffer and current position
194     // as another Stream
195     Stream(const Stream& other)
196     {
197         stream_ = other.stream_; // shallow assign
198     }
199
200     // Creates a new Stream from an existing already-initialized buffer, whose
201     // encapsulation has already been set
202     Stream(
203             unsigned char *buffer,
204             length_t size,
205             length_t offset)
206     {
207         RTIXCdrFlatData_initializeStream(
208                 &stream_,
209                 buffer,
210                 offset,
211                 size);
212     }
213
214     Stream& operator=(const Stream& other)
215     {
216         stream_ = other.stream_; // shallow assign (see copy constructor)
217         return *this;
218     }
219
220     bool skip(length_t size)
```

```
221     {
222         if (!check_size(size)) {
223             return false;
224         }
225         RTIXCdrStream_increaseCurrentPosition(&stream_, size);
226         return true;
227     }
228
229     void skip_fast(length_t size)
230     {
231         RTIXCdrStream_increaseCurrentPosition(&stream_, size);
232     }
233
234     void skip_back(length_t bytes)
235     {
236         RTIXCdrStream_increaseCurrentPosition(
237                 &stream_,
238                 -static_cast<int>(bytes));
239     }
240
241     unsigned char * buffer()
242     {
243         return reinterpret_cast<unsigned char*>(stream_._buffer);
244     }
245
246     const unsigned char * buffer() const
247     {
248         return reinterpret_cast<unsigned char*>(stream_._buffer);
249     }
250
251     unsigned char * current_position()
252     {
253         return reinterpret_cast<unsigned char*>(
254                 RTIXCdrStream_getCurrentPosition(&stream_));
255     }
256
257     void current_position(unsigned char * pos)
258     {
259         RTIXCdrStream_setCurrentPosition(
260                 &stream_,
261                 reinterpret_cast<char*>(pos));
262     }
263
264     length_t used_size() const
265     {
266         return static_cast<length_t>(
267                 RTIXCdrStream_getCurrentPosition(&stream_) - stream_._buffer);
268     }
269
270     length_t remaining_size() const
271     {
272         return total_size() - used_size();
273     }
274
275     length_t total_size() const
276     {
277         return stream_._bufferLength;
278     }
279
280     bool needs_byte_swap() const
281     {
282         return stream_._needByteSwap == RTI_XCDR_TRUE;
283     }
284
285     template <typename T>
286     bool serialize(T value)
287     {
288         return primitive_type_traits<T>::serialize(&stream_, &value);
289     }
290
291     template <typename T>
292     void serialize_fast(T value)
293     {
294         primitive_type_traits<T>::serialize_fast(&stream_, &value);
295     }
296
297     template <typename T>
298     bool serialize_no_align(T value)
299     {
300         if (!check_size(sizeof(T))) {
301             return false;
```

```
302         }
303         primitive_type_traits<T>::serialize_fast(&stream_, &value);
304         return true;
305     }
306 
307     template <typename T>
308     T deserialize_fast()
309     {
310         T value;
311         primitive_type_traits<T>::deserialize_fast(&stream_, &value);
312         return value;
313     }
314 
315     void serialize_fast(const void *array, length_t bytes)
316     {
317         RTIXCdrStream_serializeNByteFast(&stream_, array, bytes);
318     }
319 
320     char * serialize_dheader()
321     {
322         return RTIXCdrStream_serializeDHeader(&stream_);
323     }
324 
325     bool finish_dheader(char *dheader_position)
326     {
327         return RTIXCdrStream_serializeDHeaderLength(&stream_, dheader_position)
328                 == RTI_XCDR_TRUE;
329     }
330 
331     bool serialize_emheader(
332             char*& position,
333             RTIXCdrUnsignedLong parameter_id,
334             RTIXCdrUnsignedLong lc,
335             bool must_understand = false)
336     {
337         RTIXCdrBoolean failure = 0;
338         position = RTIXCdrStream_serializeV2ParameterHeader(
339                 &stream_,
340                 &failure,
341                 parameter_id,
342                 must_understand ? RTI_XCDR_TRUE : RTI_XCDR_FALSE,
343                 lc);
344 
345         return failure == RTI_XCDR_FALSE;
346     }
347 
348     void finish_emheader(char * emheader_position)
349     {
350         RTIXCdrStream_finishV2ParameterHeader(&stream_, emheader_position);
351     }
352 
353     bool align(unsigned int alignment)
354     {
355         return RTIXCdrStream_align(&stream_, (RTIXCdrAlignment) alignment)
356                 == RTI_XCDR_TRUE;
357     }
358 
359     bool check_size(length_t size) const
360     {
361         bool result = RTIXCdrStream_checkSize(&stream_, size) == RTI_XCDR_TRUE;
362         return result;
363     }
364 
365     bool need_byte_swap() const
366     {
367         return stream_._needByteSwap != 0;
368     }
369 
370     RTIXCdrStream& c_stream()
371     {
372         return stream_;
373     }
374 
375 
376     // Saves the current position of a stream so it can be restored later on if
377     // needed. This utility has two main uses:
378     //  - For Offsets: move the offset to a member, then go back
379     //  - For Builders: allow rolling back to the state before starting building
380     //      a member, in case of an error or a call to the builder's discard()
381     //
382     // For maintainability we also save the relativeBuffer, but FlatData doesn't
```

```
383     // use it.
384     class Memento {
385     public:
386         explicit Memento(Stream& stream) :
387                 stream_(stream),
388                 original_position_(stream.current_position()),
389                 relative_buffer_(stream.c_stream()._relativeBuffer)
390         {
391         }
392 
393         // Restores the saved position unless discard() has been called
394         ~Memento()
395         {
396             restore();
397         }
398 
399         // Restores the saved position unless discard() has been called
400         void restore()
401         {
402             if (original_position_ != NULL) {
403                 stream_.current_position(original_position_);
404                 stream_.c_stream()._relativeBuffer = relative_buffer_;
405             }
406         }
407 
408         // Call this to no longer restore the original position
409         unsigned char * discard()
410         {
411             unsigned char *pos = original_position_;
412             original_position_ = NULL;
413             return pos;
414         }
415 
416     private:
417         Stream& stream_;
418         unsigned char *original_position_;
419         char *relative_buffer_;
420     };
421 
422 
423 private:
424     RTIXCdrStream stream_;
425 };
426 
427 } } // namespace rti::xcdr
428 
429 #endif // RTI_XCDR_STREAM_HPP_
```

Chapter 11

Example Documentation

11.1 HelloWorld.idl

11.1.1 IDL Type Description

The data type to be disseminated by RTI Connext is described in language-independent IDL. The IDL file is input to rtiddsgen (see the `Code Generator User's Manual` for more information), which produces the following files.

The programming language specific type representation of the type **Foo** (p. 1916) = HelloWorld, for use in the application code.

- HelloWorld.h
- HelloWorld.cxx

User Data Type Support (p. 73) types as required by the DDS specification for use in the application code.

- HelloWorldSupport.h
- HelloWorldSupport.cxx

Methods required by the RTI Connext implementation. These contains the auto-generated methods for serializing and deserializing the type.

- HelloWorldPlugin.h
- HelloWorldPlugin.cxx

11.1.1.1 HelloWorld.idl

```
struct HelloWorld {
    string<128> msg;
};
```

11.2 HelloWorld.cxx

11.2.1 Programming Language Type Description

The following programming language specific type representation is generated by rtiddsgen (see the `Code Generator User's Manual` for more information) for use in application code, where:

- **Foo** (p. 1916) = HelloWorld
- **FooSeq** (p. 1965) = HelloWorldSeq

11.2.1.1 HelloWorld.h

```
/*
WARNING: THIS FILE IS AUTO-GENERATED. DO NOT MODIFY.

This file was generated from HelloWorld.idl
using RTI Code Generator (rtiddsgen) version 4.7.0.
The rtiddsgen tool is part of the RTI Connext DDS distribution.
For more information, type 'rtiddsgen -help' at a command shell
or consult the Code Generator User's Manual.
*/
#ifndef HelloWorld_1436885487_h
#define HelloWorld_1436885487_h
#ifndef NDDS_STANDALONE_TYPE
#ifndef ndds_cpp_h
#include "ndds/ndds_cpp.h"
#endif
#include "rti/xcdr/Interpreter.hpp"
#include "cdr/cdr_typeCode.h"
#else
#include "ndds_standalone_type.h"
#endif
extern "C" {
    extern const char *HelloWorldTYPENAME;
}
struct HelloWorldSeq;
#ifndef NDDS_STANDALONE_TYPE
class HelloWorldTypeSupport;
class HelloWorldDataWriter;
class HelloWorldDataReader;
#endif
class HelloWorld
{
  public:
    typedef struct HelloWorldSeq Seq;
    #ifndef NDDS_STANDALONE_TYPE
    typedef HelloWorldTypeSupport TypeSupport;
    typedef HelloWorldDataWriter DataWriter;
    typedef HelloWorldDataReader DataReader;
    #endif
    DDS_Char * msg;
};
#if defined(NDDS_USER_DLL_EXPORT) && defined(RTI_WIN32)
#undef NDDSUSERDllExport
#define NDDSUSERDllExport __declspec(dllexport)
#endif
#if !defined(RTI_WIN32) && defined(NDDS_USER_SYMBOL_EXPORT)
#undef NDDSUSERDllExport
#define NDDSUSERDllExport __attribute__((visibility("default")))
#endif
#ifndef NDDS_STANDALONE_TYPE
NDDSUSERDllExport DDS_TypeCode * HelloWorld_get_typecode(void); /* Type code */
NDDSUSERDllExport RTIXCdrTypePlugin *HelloWorld_get_type_plugin_info(void);
NDDSUSERDllExport RTIXCdrSampleAccessInfo *HelloWorld_get_sample_access_info(void);
NDDSUSERDllExport RTIXCdrSampleAccessInfo *HelloWorld_get_sample_seq_access_info(void);
#endif
DDS_SEQUENCE(HelloWorldSeq, HelloWorld);
NDDSUSERDllExport
RTIBool HelloWorld_initialize(
    HelloWorld* self);
```

```
NDDSUSERDllExport
RTIBool HelloWorld_initialize_ex(
    HelloWorld* self,RTIBool allocatePointers,RTIBool allocateMemory);
NDDSUSERDllExport
RTIBool HelloWorld_initialize_w_params(
    HelloWorld* self,
    const struct DDS_TypeAllocationParams_t * allocParams);
NDDSUSERDllExport
RTIBool HelloWorld_finalize_w_return(
    HelloWorld* self);
NDDSUSERDllExport
void HelloWorld_finalize(
    HelloWorld* self);
NDDSUSERDllExport
void HelloWorld_finalize_ex(
    HelloWorld* self,RTIBool deletePointers);
NDDSUSERDllExport
void HelloWorld_finalize_w_params(
    HelloWorld* self,
    const struct DDS_TypeDeallocationParams_t * deallocParams);
NDDSUSERDllExport
void HelloWorld_finalize_optional_members(
    HelloWorld* self, RTIBool deletePointers);
NDDSUSERDllExport
RTIBool HelloWorld_copy(
    HelloWorld* dst,
    const HelloWorld* src);
#if defined(NDDS_USER_DLL_EXPORT) || defined(NDDS_USER_SYMBOL_EXPORT)
#undef NDDSUSERDllExport
#define NDDSUSERDllExport
#endif
#ifndef NDDS_STANDALONE_TYPE
namespace rti {
    namespace xcdr {
        template <>
        struct type_code< HelloWorld> {
            static const RTIXCdrTypeCode * get();
        };
    }
}
#endif
#endif /* HelloWorld */
```

11.2.1.2 HelloWorld.cxx

```
/*
WARNING: THIS FILE IS AUTO-GENERATED. DO NOT MODIFY.

This file was generated from HelloWorld.idl
using RTI Code Generator (rtiddsgen) version 4.7.0.
The rtiddsgen tool is part of the RTI Connext DDS distribution.
For more information, type 'rtiddsgen -help' at a command shell
or consult the Code Generator User's Manual.
*/
#ifndef NDDS_STANDALONE_TYPE
#ifndef ndds_cpp_h
#include "ndds/ndds_cpp.h"
#endif
#ifndef dds_c_log_impl_h
#include "dds_c/dds_c_log_impl.h"
#endif
#ifndef dds_c_log_infrastructure_h
#include "dds_c/dds_c_infrastructure_impl.h"
#endif
#ifndef cdr_type_h
#include "cdr/cdr_type.h"
#endif
#include "osapi/osapi_atomic.h"
#else
#include "ndds_standalone_type.h"
#endif
#include "HelloWorld.h"
#ifndef NDDS_STANDALONE_TYPE
#include "HelloWorldPlugin.h"
#endif
#include <new>
/* ========================================================================= */
const char *HelloWorldTYPENAME = "HelloWorld";
```

```
#ifndef NDDS_STANDALONE_TYPE
DDS_TypeCode * HelloWorld_get_typecode(void)
{
    static RTI_ATOMIC(RTIBool) is_initialized;
    static DDS_TypeCode HelloWorld_g_tc_msg_string = DDS_INITIALIZE_STRING_TYPECODE((128L));
    static DDS_TypeCode_Member HelloWorld_g_tc_members[1]=
    {
        {
            (char *)"msg",/* Member name */
            {
                0,/* Representation ID */
                DDS_BOOLEAN_FALSE,/* Is a pointer? */
                -1, /* Bitfield bits */
                NULL/* Member type code is assigned later */
            },
            0, /* Ignored */
            0, /* Ignored */
            0, /* Ignored */
            NULL, /* Ignored */
            RTI_CDR_REQUIRED_MEMBER, /* Is a key? */
            DDS_PUBLIC_MEMBER,/* Member visibility */
            RTICdrTypeCodeAnnotations_INITIALIZER
        }
    };
    static DDS_TypeCode HelloWorld_g_tc =
    {{
            DDS_TK_STRUCT, /* Kind */
            DDS_BOOLEAN_FALSE, /* Ignored */
            -1, /*Ignored*/
            (char *)"HelloWorld", /* Name */
            NULL, /* Ignored */
            0, /* Ignored */
            0, /* Ignored */
            NULL, /* Ignored */
            1, /* Number of members */
            HelloWorld_g_tc_members, /* Members */
            DDS_VM_NONE, /* Ignored */
            RTICdrTypeCodeAnnotations_INITIALIZER,
            DDS_BOOLEAN_TRUE, /* _isCopyable */
            NULL, /* _sampleAccessInfo: assigned later */
            NULL /* _typePlugin: assigned later */
        }}; /* Type code for HelloWorld*/
    if (RTIOsapiAtomic_load32(&is_initialized, RTI_OSAPI_ATOMIC_MEMORY_ORDER_ACQUIRE)) {
        return &HelloWorld_g_tc;
    }
    HelloWorld_g_tc._data._annotations._allowedDataRepresentationMask = 5;
    HelloWorld_g_tc_members[0]._representation._typeCode =  (RTICdrTypeCode *)&HelloWorld_g_tc_msg_string;
    /* Initialize the values for member annotations. */
    HelloWorld_g_tc_members[0]._annotations._defaultValue._d = RTI_XCDR_TK_STRING;
    HelloWorld_g_tc_members[0]._annotations._defaultValue._u.string_value = (DDS_Char *) "";
    HelloWorld_g_tc._data._sampleAccessInfo =
    HelloWorld_get_sample_access_info();
    HelloWorld_g_tc._data._typePlugin =
    HelloWorld_get_type_plugin_info();
    RTIOsapiAtomic_store32(
        &is_initialized,
        RTI_TRUE,
        RTI_OSAPI_ATOMIC_MEMORY_ORDER_RELEASE);
    return &HelloWorld_g_tc;
}
#define TSeq HelloWorldSeq
#define T HelloWorld
#include "dds_cpp/generic/dds_cpp_data_TInterpreterSupport.gen"
#undef T
#undef TSeq
RTIXCdrSampleAccessInfo *HelloWorld_get_sample_seq_access_info()
{
    static RTIXCdrSampleAccessInfo HelloWorld_g_seqSampleAccessInfo = {
        RTI_XCDR_TYPE_BINDING_CPP, \
        {sizeof(HelloWorldSeq),0,0,0}, \
        RTI_XCDR_FALSE, \
        DDS_Sequence_get_member_value_pointer, \
        HelloWorldSeq_set_member_element_count, \
        NULL, \
        NULL, \
        NULL \
    };
    return &HelloWorld_g_seqSampleAccessInfo;
}
RTIXCdrSampleAccessInfo *HelloWorld_get_sample_access_info()
{
```

```
    static RTI_ATOMIC(RTIBool) is_initialized;
    HelloWorld *sample;
    static RTIXCdrMemberAccessInfo HelloWorld_g_memberAccessInfos[1] =
    {RTIXCdrMemberAccessInfo_INITIALIZER};
    static RTIXCdrSampleAccessInfo HelloWorld_g_sampleAccessInfo =
    RTIXCdrSampleAccessInfo_INITIALIZER;
    if (RTIOsapiAtomic_load32(
        &is_initialized,
        RTI_OSAPI_ATOMIC_MEMORY_ORDER_ACQUIRE)) {
        return (RTIXCdrSampleAccessInfo*) &HelloWorld_g_sampleAccessInfo;
    }
    RTIXCdrHeap_allocateStruct(
        &sample,
        HelloWorld);
    if (sample == NULL) {
        return NULL;
    }
    HelloWorld_g_memberAccessInfos[0].bindingMemberValueOffset[0] =
    (RTIXCdrUnsignedLong) ((char *)&sample->msg - (char *)sample);
    HelloWorld_g_sampleAccessInfo.memberAccessInfos =
    HelloWorld_g_memberAccessInfos;
    {
        size_t candidateTypeSize = sizeof(HelloWorld);
        if (candidateTypeSize > RTIXCdrLong_MAX) {
            HelloWorld_g_sampleAccessInfo.typeSize[0] =
            RTIXCdrLong_MAX;
        } else {
            HelloWorld_g_sampleAccessInfo.typeSize[0] =
            (RTIXCdrUnsignedLong) candidateTypeSize;
        }
    }
    HelloWorld_g_sampleAccessInfo.useGetMemberValueOnlyWithRef =
    RTI_XCDR_TRUE;
    HelloWorld_g_sampleAccessInfo.getMemberValuePointerFcn =
    HelloWorld_get_member_value_pointer;
    HelloWorld_g_sampleAccessInfo.languageBinding =
    RTI_XCDR_TYPE_BINDING_CPP ;
    RTIXCdrHeap_freeStruct(sample);
    RTIOsapiAtomic_store32(
        &is_initialized,
        RTI_TRUE,
        RTI_OSAPI_ATOMIC_MEMORY_ORDER_RELEASE);
    return (RTIXCdrSampleAccessInfo*) &HelloWorld_g_sampleAccessInfo;
}
RTIXCdrTypePlugin *HelloWorld_get_type_plugin_info()
{
    static RTIXCdrTypePlugin HelloWorld_g_typePlugin =
    {
        NULL, /* serialize */
        NULL, /* serialize_key */
        NULL, /* deserialize_sample */
        NULL, /* deserialize_key_sample */
        NULL, /* skip */
        NULL, /* get_serialized_sample_size */
        NULL, /* get_serialized_sample_max_size_ex */
        NULL, /* get_serialized_key_max_size_ex */
        NULL, /* get_serialized_sample_min_size */
        NULL, /* serialized_sample_to_key */
        (RTIXCdrTypePluginInitializeSampleFunction)
        HelloWorld_initialize_ex,
        NULL,
        (RTIXCdrTypePluginFinalizeSampleFunction)
        HelloWorld_finalize_w_return,
        NULL,
        NULL
    };
    return &HelloWorld_g_typePlugin;
}
#endif
RTIBool HelloWorld_initialize(
    HelloWorld* sample)
{
    return HelloWorld_initialize_ex(
        sample,
        RTI_TRUE,
        RTI_TRUE);
}
RTIBool HelloWorld_initialize_w_params(
    HelloWorld *sample,
    const struct DDS_TypeAllocationParams_t *allocParams)
{
```

```
    if (sample == NULL) {
        return RTI_FALSE;
    }
    if (allocParams == NULL) {
        return RTI_FALSE;
    }
    if (allocParams->allocate_memory) {
        sample->msg = DDS_String_alloc((128L));
        if (sample->msg != NULL) {
            RTIOsapiUtility_unusedReturnValue(
                RTICdrType_copyStringEx(
                    &sample->msg,
                    "",
                    (128L),
                    RTI_FALSE),
                    RTIBool);
        }
        if (sample->msg == NULL) {
            return RTI_FALSE;
        }
    } else {
        if (sample->msg != NULL) {
            RTIOsapiUtility_unusedReturnValue(
                RTICdrType_copyStringEx(
                    &sample->msg,
                    "",
                    (128L),
                    RTI_FALSE),
                    RTIBool);
            if (sample->msg == NULL) {
                return RTI_FALSE;
            }
        }
    }
    return RTI_TRUE;
}
RTIBool HelloWorld_initialize_ex(
    HelloWorld *sample,
    RTIBool allocatePointers,
    RTIBool allocateMemory)
{
    struct DDS_TypeAllocationParams_t allocParams =
    DDS_TYPE_ALLOCATION_PARAMS_DEFAULT;
    allocParams.allocate_pointers =  (DDS_Boolean)allocatePointers;
    allocParams.allocate_memory = (DDS_Boolean)allocateMemory;
    return HelloWorld_initialize_w_params(
        sample,
        &allocParams);
}
RTIBool HelloWorld_finalize_w_return(
    HelloWorld* sample)
{
    HelloWorld_finalize_ex(sample, RTI_TRUE);
    return RTI_TRUE;
}
void HelloWorld_finalize(
    HelloWorld* sample)
{
    HelloWorld_finalize_ex(
        sample,
        RTI_TRUE);
}
void HelloWorld_finalize_ex(
    HelloWorld *sample,
    RTIBool deletePointers)
{
    struct DDS_TypeDeallocationParams_t deallocParams =
    DDS_TYPE_DEALLOCATION_PARAMS_DEFAULT;
    if (sample==NULL) {
        return;
    }
    deallocParams.delete_pointers = (DDS_Boolean)deletePointers;
    HelloWorld_finalize_w_params(
        sample,
        &deallocParams);
}
void HelloWorld_finalize_w_params(
    HelloWorld *sample,
    const struct DDS_TypeDeallocationParams_t *deallocParams)
{
    if (sample==NULL) {
```

```
        return;
    }
    if (deallocParams == NULL) {
        return;
    }
    if (sample->msg != NULL) {
        DDS_String_free(sample->msg);
        sample->msg=NULL;
    }
}
void HelloWorld_finalize_optional_members(
    HelloWorld* sample, RTIBool deletePointers)
{
    struct DDS_TypeDeallocationParams_t deallocParamsTmp =
    DDS_TYPE_DEALLOCATION_PARAMS_DEFAULT;
    struct DDS_TypeDeallocationParams_t * deallocParams =
    &deallocParamsTmp;
    if (sample==NULL) {
        return;
    }
    RTIOsapiUtility_unusedParameter(deallocParams);
    deallocParamsTmp.delete_pointers = (DDS_Boolean)deletePointers;
    deallocParamsTmp.delete_optional_members = DDS_BOOLEAN_TRUE;
}
RTIBool HelloWorld_copy(
    HelloWorld* dst,
    const HelloWorld* src)
{
    try {
        if (dst == NULL || src == NULL) {
            return RTI_FALSE;
        }
        if (!RTICdrType_copyStringEx (
            &dst->msg
            ,
            src->msg,
            (128L) + 1,
            RTI_FALSE)){
            return RTI_FALSE;
        }
        return RTI_TRUE;
    } catch (const std::bad_alloc&) {
        return RTI_FALSE;
    }
}
#define T HelloWorld
#define TSeq HelloWorldSeq
#define T_initialize_w_params HelloWorld_initialize_w_params
#define T_finalize_w_params   HelloWorld_finalize_w_params
#define T_copy       HelloWorld_copy
#ifndef NDDS_STANDALONE_TYPE
#include "dds_c/generic/dds_c_sequence_TSeq.gen"
#include "dds_cpp/generic/dds_cpp_sequence_TSeq.gen"
#else
#include "dds_c_sequence_TSeq.gen"
#include "dds_cpp_sequence_TSeq.gen"
#endif
#undef T_copy
#undef T_finalize_w_params
#undef T_initialize_w_params
#undef TSeq
#undef T
#ifndef NDDS_STANDALONE_TYPE
namespace rti {
    namespace xcdr {
        const RTIXCdrTypeCode * type_code< HelloWorld>::get()
        {
            return (const RTIXCdrTypeCode *) HelloWorld_get_typecode();
        }
    }
}
#endif
```

11.3 HelloWorldSupport.cxx

11.3.1 User Data Type Support

Files generated by rtiddsgen (see the `Code Generator User's Manual` for more information) that implement the type specific APIs required by the DDS specification, as described in the **User Data Type Support** (p. 73), where:

- **FooTypeSupport** (p. 1979) = HelloWorldTypeSupport
- **FooDataWriter** (p. 1944) = HelloWorldDataWriter
- **FooDataReader** (p. 1917) = HelloWorldDataReader

11.3.1.1 HelloWorldSupport.h

```
/*
WARNING: THIS FILE IS AUTO-GENERATED. DO NOT MODIFY.

This file was generated from HelloWorld.idl
using RTI Code Generator (rtiddsgen) version 4.7.0.
The rtiddsgen tool is part of the RTI Connext DDS distribution.
For more information, type 'rtiddsgen -help' at a command shell
or consult the Code Generator User's Manual.
*/
#ifndef HelloWorldSupport_1436885487_h
#define HelloWorldSupport_1436885487_h
/* Uses */
#include "HelloWorld.h"
#ifndef ndds_cpp_h
#include "ndds/ndds_cpp.h"
#endif
#if (defined(RTI_WIN32) || defined (RTI_WINCE) || defined(RTI_INTIME)) && defined(NDDS_USER_DLL_EXPORT)
class __declspec(dllimport) DDSTypeSupport;
class __declspec(dllimport) DDSDataWriter;
class __declspec(dllimport) DDSDataReader;
#endif
/* ========================================================================= */
#if defined(NDDS_USER_DLL_EXPORT) && defined(RTI_WIN32)
#undef NDDSUSERDllExport
#define NDDSUSERDllExport __declspec(dllexport)
#endif
#if !defined(RTI_WIN32) && defined(NDDS_USER_SYMBOL_EXPORT)
#undef NDDSUSERDllExport
#define NDDSUSERDllExport __attribute__((visibility("default")))
#endif
DDS_TYPESUPPORT_CPP(
    HelloWorldTypeSupport,
    HelloWorld);
#define RTI_ENABLE_TDATAWRITER_DATA_CONSTRUCTOR_METHODS
DDS_DATAWRITER_WITH_DATA_CONSTRUCTOR_METHODS_CPP(HelloWorldDataWriter, HelloWorld);
#undef RTI_ENABLE_TDATAWRITER_DATA_CONSTRUCTOR_METHODS
#define RTI_ENABLE_TDATAREADER_DATA_CONSISTENCY_CHECK_METHOD
DDS_DATAREADER_W_DATA_CONSISTENCY_CHECK(HelloWorldDataReader, HelloWorldSeq, HelloWorld);
#undef RTI_ENABLE_TDATAREADER_DATA_CONSISTENCY_CHECK_METHOD
#if defined(NDDS_USER_DLL_EXPORT) || defined(NDDS_USER_SYMBOL_EXPORT)
#undef NDDSUSERDllExport
#define NDDSUSERDllExport
#endif
#endif  /* HelloWorldSupport_1436885487_h */
```

11.3.1.2 HelloWorldSupport.cxx

```
/*
WARNING: THIS FILE IS AUTO-GENERATED. DO NOT MODIFY.

This file was generated from HelloWorld.idl
using RTI Code Generator (rtiddsgen) version 4.7.0.
The rtiddsgen tool is part of the RTI Connext DDS distribution.
```

```
For more information, type 'rtiddsgen -help' at a command shell
or consult the Code Generator User's Manual.
*/
#include "HelloWorldSupport.h"
#include "HelloWorldPlugin.h"
#ifndef dds_c_log_impl_h
#include "dds_c/dds_c_log_impl.h"
#endif
/* ========================================================================= */
/* ----------------------------------------------------------------- */
/* DDSDataWriter
*/
/* Requires */
#define TTYPENAME   HelloWorldTYPENAME
/* Defines */
#define TDataWriter HelloWorldDataWriter
#define TData       HelloWorld
#define RTI_ENABLE_TDATAWRITER_DATA_CONSTRUCTOR_METHODS
#include "dds_cpp/generic/dds_cpp_data_TDataWriter.gen"
#undef RTI_ENABLE_TDATAWRITER_DATA_CONSTRUCTOR_METHODS
#undef TDataWriter
#undef TData
#undef TTYPENAME
/* ----------------------------------------------------------------- */
/* DDSDataReader
*/
/* Requires */
#define TTYPENAME   HelloWorldTYPENAME
/* Defines */
#define TDataReader HelloWorldDataReader
#define TDataSeq    HelloWorldSeq
#define TData       HelloWorld
#define RTI_ENABLE_TDATAREADER_DATA_CONSISTENCY_CHECK_METHOD
#include "dds_cpp/generic/dds_cpp_data_TDataReader.gen"
#undef RTI_ENABLE_TDATAREADER_DATA_CONSISTENCY_CHECK_METHOD
#undef TDataReader
#undef TDataSeq
#undef TData
#undef TTYPENAME
/* ----------------------------------------------------------------- */
/* TypeSupport

«IMPLEMENTATION »

Requires:  TTYPENAME,
TPlugin_new
TPlugin_delete
Defines:   TTypeSupport, TData, TDataReader, TDataWriter
*/
/* Requires */
#define TTYPENAME    HelloWorldTYPENAME
#define TPlugin_new  HelloWorldPlugin_new
#define TPlugin_delete  HelloWorldPlugin_delete
/* Defines */
#define TTypeSupport HelloWorldTypeSupport
#define TData        HelloWorld
#define TDataReader  HelloWorldDataReader
#define TDataWriter  HelloWorldDataWriter
#define TGENERATE_SER_CODE
#ifndef NDDS_STANDALONE_TYPE
#define TGENERATE_TYPECODE
#endif
#include "dds_cpp/generic/dds_cpp_data_TTypeSupport.gen"
#undef TTypeSupport
#undef TData
#undef TDataReader
#undef TDataWriter
#ifndef NDDS_STANDALONE_TYPE
#undef TGENERATE_TYPECODE
#endif
#undef TGENERATE_SER_CODE
#undef TTYPENAME
#undef TPlugin_new
#undef TPlugin_delete
```

11.4 HelloWorldPlugin.cxx

11.4.1 RTI Connext Implementation Support

Files generated by rtiddsgen (see the `Code Generator User's Manual` for more information) that provided methods for type specific serialization and deserialization, to support the RTI Connext implementation.

11.4.1.1 HelloWorldPlugin.h

```
/*
WARNING: THIS FILE IS AUTO-GENERATED. DO NOT MODIFY.

This file was generated from HelloWorld.idl
using RTI Code Generator (rtiddsgen) version 4.7.0.
The rtiddsgen tool is part of the RTI Connext DDS distribution.
For more information, type 'rtiddsgen -help' at a command shell
or consult the Code Generator User's Manual.
*/
#ifndef HelloWorldPlugin_1436885487_h
#define HelloWorldPlugin_1436885487_h
#include "HelloWorld.h"
struct RTICdrStream;
#ifndef pres_typePlugin_h
#include "pres/pres_typePlugin.h"
#endif
#if defined(NDDS_USER_DLL_EXPORT) && defined(RTI_WIN32)
#undef NDDSUSERDllExport
#define NDDSUSERDllExport __declspec(dllexport)
#endif
#if !defined(RTI_WIN32) && defined(NDDS_USER_SYMBOL_EXPORT)
#undef NDDSUSERDllExport
#define NDDSUSERDllExport __attribute__((visibility("default")))
#endif
extern "C"{
    #define HelloWorldPlugin_get_sample PRESTypePluginDefaultEndpointData_getSample
    #define HelloWorldPlugin_get_buffer PRESTypePluginDefaultEndpointData_getBuffer
    #define HelloWorldPlugin_return_buffer PRESTypePluginDefaultEndpointData_returnBuffer
    #define HelloWorldPlugin_create_sample PRESTypePluginDefaultEndpointData_createSample
    #define HelloWorldPlugin_destroy_sample PRESTypePluginDefaultEndpointData_deleteSample
    /* --------------------------------------------------------------------------------------
    Support functions:
    * -------------------------------------------------------------------------------------- */
    NDDSUSERDllExport extern HelloWorld*
    HelloWorldPluginSupport_create_data_w_params(
        const struct DDS_TypeAllocationParams_t * alloc_params);
    NDDSUSERDllExport extern HelloWorld*
    HelloWorldPluginSupport_create_data_ex(RTIBool allocate_pointers);
    NDDSUSERDllExport extern HelloWorld*
    HelloWorldPluginSupport_create_data(void);
    NDDSUSERDllExport extern RTIBool
    HelloWorldPluginSupport_copy_data(
        HelloWorld *out,
        const HelloWorld *in);
    NDDSUSERDllExport extern void
    HelloWorldPluginSupport_destroy_data_w_params(
        HelloWorld *sample,
        const struct DDS_TypeDeallocationParams_t * dealloc_params);
    NDDSUSERDllExport extern void
    HelloWorldPluginSupport_destroy_data_ex(
        HelloWorld *sample,RTIBool deallocate_pointers);
    NDDSUSERDllExport extern void
    HelloWorldPluginSupport_destroy_data(
        HelloWorld *sample);
    NDDSUSERDllExport extern void
    HelloWorldPluginSupport_print_data(
        const HelloWorld *sample,
        const char *desc,
        unsigned int indent);
    /* ----------------------------------------------------------------------------
    Callback functions:
    * ---------------------------------------------------------------------------- */
    NDDSUSERDllExport extern PRESTypePluginParticipantData
    HelloWorldPlugin_on_participant_attached(
```

```
    void *registration_data,
    const struct PRESTypePluginParticipantInfo *participant_info,
    RTIBool top_level_registration,
    void *container_plugin_context,
    RTICdrTypeCode *typeCode);
NDDSUSERDllExport extern void
HelloWorldPlugin_on_participant_detached(
    PRESTypePluginParticipantData participant_data);
NDDSUSERDllExport extern PRESTypePluginEndpointData
HelloWorldPlugin_on_endpoint_attached(
    PRESTypePluginParticipantData participant_data,
    const struct PRESTypePluginEndpointInfo *endpoint_info,
    RTIBool top_level_registration,
    void *container_plugin_context);
NDDSUSERDllExport extern void
HelloWorldPlugin_on_endpoint_detached(
    PRESTypePluginEndpointData endpoint_data);
NDDSUSERDllExport extern void
HelloWorldPlugin_return_sample(
    PRESTypePluginEndpointData endpoint_data,
    HelloWorld *sample,
    void *handle);
NDDSUSERDllExport extern RTIBool
HelloWorldPlugin_copy_sample(
    PRESTypePluginEndpointData endpoint_data,
    HelloWorld *out,
    const HelloWorld *in);
/* ----------------------------------------------------------------------------
(De)Serialize functions:
* ------------------------------------------------------------------------- */
NDDSUSERDllExport extern RTIBool
HelloWorldPlugin_serialize_to_cdr_buffer(
    char * buffer,
    unsigned int * length,
    const HelloWorld *sample);
NDDSUSERDllExport extern RTIBool
HelloWorldPlugin_serialize_to_cdr_buffer_ex(
    char *buffer,
    unsigned int *length,
    const HelloWorld *sample,
    DDS_DataRepresentationId_t representation);
NDDSUSERDllExport extern RTIBool
HelloWorldPlugin_deserialize_from_cdr_buffer(
    HelloWorld *sample,
    const char * buffer,
    unsigned int length);
#if !defined (NDDS_STANDALONE_TYPE)
NDDSUSERDllExport extern DDS_ReturnCode_t
HelloWorldPlugin_data_to_string(
    const HelloWorld *sample,
    char *str,
    DDS_UnsignedLong *str_size,
    const struct DDS_PrintFormatProperty *property);
#endif
NDDSUSERDllExport extern unsigned int
HelloWorldPlugin_get_serialized_sample_max_size(
    PRESTypePluginEndpointData endpoint_data,
    RTIBool include_encapsulation,
    RTIEncapsulationId encapsulation_id,
    unsigned int current_alignment);
/* ----------------------------------------------------------------------------
Key Management functions:
* ------------------------------------------------------------------------- */
NDDSUSERDllExport extern PRESTypePluginKeyKind
HelloWorldPlugin_get_key_kind(void);
NDDSUSERDllExport extern unsigned int
HelloWorldPlugin_get_serialized_key_max_size(
    PRESTypePluginEndpointData endpoint_data,
    RTIBool include_encapsulation,
    RTIEncapsulationId encapsulation_id,
    unsigned int current_alignment);
NDDSUSERDllExport extern unsigned int
HelloWorldPlugin_get_serialized_key_max_size_for_keyhash(
    PRESTypePluginEndpointData endpoint_data,
    RTIEncapsulationId encapsulation_id,
    unsigned int current_alignment);
NDDSUSERDllExport extern RTIBool
HelloWorldPlugin_deserialize_key(
    PRESTypePluginEndpointData endpoint_data,
    HelloWorld ** sample,
    RTIBool * drop_sample,
```

```
        struct RTICdrStream *cdrStream,
        RTIBool deserialize_encapsulation,
        RTIBool deserialize_key,
        void *endpoint_plugin_qos);
    NDDSUSERDllExport extern
    struct RTIXCdrInterpreterPrograms * HelloWorldPlugin_get_programs(void);
    /* Plugin Functions */
    NDDSUSERDllExport extern struct PRESTypePlugin*
    HelloWorldPlugin_new(void);
    NDDSUSERDllExport extern void
    HelloWorldPlugin_delete(struct PRESTypePlugin *);
}
#if defined(NDDS_USER_DLL_EXPORT) || defined(NDDS_USER_SYMBOL_EXPORT)
#undef NDDSUSERDllExport
#define NDDSUSERDllExport
#endif
#endif /* HelloWorldPlugin_1436885487_h */
```

11.4.1.2 HelloWorldPlugin.cxx

```
/*
WARNING: THIS FILE IS AUTO-GENERATED. DO NOT MODIFY.

This file was generated from HelloWorld.idl
using RTI Code Generator (rtiddsgen) version 4.7.0.
The rtiddsgen tool is part of the RTI Connext DDS distribution.
For more information, type 'rtiddsgen -help' at a command shell
or consult the Code Generator User's Manual.
*/
#include <string.h>
#ifndef ndds_cpp_h
#include "ndds/ndds_cpp.h"
#endif
#ifndef cdr_type_h
#include "cdr/cdr_type.h"
#endif
#ifndef cdr_type_object_h
#include "cdr/cdr_typeObject.h"
#endif
#ifndef cdr_encapsulation_h
#include "cdr/cdr_encapsulation.h"
#endif
#ifndef cdr_stream_h
#include "cdr/cdr_stream.h"
#endif
#include "xcdr/xcdr_interpreter.h"
#include "xcdr/xcdr_stream.h"
#ifndef cdr_log_h
#include "cdr/cdr_log.h"
#endif
#ifndef pres_typePlugin_h
#include "pres/pres_typePlugin.h"
#endif
#include "dds_c/dds_c_typecode_impl.h"
#define RTI_CDR_CURRENT_SUBMODULE RTI_CDR_SUBMODULE_MASK_STREAM
#include <new>
#include "HelloWorldPlugin.h"
/* ------------------------------------------------------------------------
*  Type HelloWorld
* ------------------------------------------------------------------------ */
/* ----------------------------------------------------------------------------
Support functions:
* ------------------------------------------------------------------------ */
HelloWorld*
HelloWorldPluginSupport_create_data_w_params(
    const struct DDS_TypeAllocationParams_t * alloc_params)
{
    HelloWorld *sample = NULL;
    if (alloc_params == NULL) {
        return NULL;
    } else if(!alloc_params->allocate_memory) {
        RTICdrLog_exception(&RTI_CDR_LOG_TYPE_OBJECT_NOT_ASSIGNABLE_ss,
        "alloc_params->allocate_memory","false");
        return NULL;
    }
    sample = new (std::nothrow) HelloWorld();
    if (sample == NULL) {
        return NULL;
```

```
    }
    if (!HelloWorld_initialize_w_params(sample,alloc_params)) {
        struct DDS_TypeDeallocationParams_t deallocParams =
        DDS_TYPE_DEALLOCATION_PARAMS_DEFAULT;
        deallocParams.delete_pointers = alloc_params->allocate_pointers;
        deallocParams.delete_optional_members = alloc_params->allocate_pointers;
        /* Coverity reports a possible uninit_use_in_call that will happen if the
        allocation fails. But if the allocation fails then sample == null and
        the method will return before reach this point.*/
        /* Coverity reports a possible overwrite_var on the members of the sample.
        It is a false positive since all the pointers are freed before assigning
        null to them. */
        /* coverity[uninit_use_in_call : FALSE] */
        /* coverity[overwrite_var : FALSE] */
        HelloWorld_finalize_w_params(sample, &deallocParams);
        /* Coverity reports a possible leaked_storage on the sample members when
        freeing sample. It is a false positive since all the members' memory
        is freed in the call "HelloWorld_finalize_ex" */
        /* coverity[leaked_storage : FALSE] */
        delete  sample;
        sample=NULL;
    }
    return sample;
}
HelloWorld *
HelloWorldPluginSupport_create_data_ex(RTIBool allocate_pointers)
{
    HelloWorld *sample = NULL;
    sample = new (std::nothrow) HelloWorld();
    if(sample == NULL) {
        return NULL;
    }
    /* coverity[example_checked : FALSE] */
    if (!HelloWorld_initialize_ex(sample,allocate_pointers, RTI_TRUE)) {
        /* Coverity reports a possible uninit_use_in_call that will happen if the
        new fails. But if new fails then sample == null and the method will
        return before reach this point. */
        /* Coverity reports a possible overwrite_var on the members of the sample.
        It is a false positive since all the pointers are freed before assigning
        null to them. */
        /* coverity[uninit_use_in_call : FALSE] */
        /* coverity[overwrite_var : FALSE] */
        HelloWorld_finalize_ex(sample, RTI_TRUE);
        /* Coverity reports a possible leaked_storage on the sample members when
        freeing sample. It is a false positive since all the members' memory
        is freed in the call "HelloWorld_finalize_ex" */
        /* coverity[leaked_storage : FALSE] */
        delete  sample;
        sample=NULL;
    }
    return sample;
}
void *
HelloWorldPluginSupport_create_dataI(void)
{
    return HelloWorldPluginSupport_create_data_ex(RTI_TRUE);
}
HelloWorld *
HelloWorldPluginSupport_create_data(void)
{
    return (HelloWorld *) HelloWorldPluginSupport_create_dataI();
}
void
HelloWorldPluginSupport_destroy_data_w_params(
    HelloWorld *sample,
    const struct DDS_TypeDeallocationParams_t * dealloc_params) {
    HelloWorld_finalize_w_params(sample,dealloc_params);
    delete  sample;
}
void
HelloWorldPluginSupport_destroy_data_ex(
    HelloWorld *sample,RTIBool deallocate_pointers) {
    HelloWorld_finalize_ex(sample,deallocate_pointers);
    delete  sample;
}
void
HelloWorldPluginSupport_destroy_data(
    HelloWorld *sample) {
    HelloWorldPluginSupport_destroy_data_ex(sample,RTI_TRUE);
}
void
```

```
HelloWorldPluginSupport_destroy_dataI(
    void *sample)
{
    HelloWorldPluginSupport_destroy_data((HelloWorld *) sample);
}
RTIBool
HelloWorldPluginSupport_copy_data(
    HelloWorld *dst,
    const HelloWorld *src)
{
    return HelloWorld_copy(dst,(const HelloWorld*) src);
}
void
HelloWorldPluginSupport_print_data(
    const HelloWorld *sample,
    const char *desc,
    unsigned int indent_level)
{
    RTICdrType_printIndent(indent_level);
    if (desc != NULL) {
        RTILogParamString_printPlain("%s:\n", desc);
    } else {
        RTILogParamString_printPlain("\n");
    }
    if (sample == NULL) {
        RTILogParamString_printPlain("NULL\n");
        return;
    }
    if (sample->msg==NULL) {
        RTICdrType_printString(
            NULL,
            "msg",
            RTIOsapiUtility_uInt32Plus1(indent_level));
    } else {
        RTICdrType_printString(
            sample->msg,
            "msg",
            RTIOsapiUtility_uInt32Plus1(indent_level));
    }
}
/* ------------------------------------------------------------------------
Callback functions:
* ------------------------------------------------------------------------ */
PRESTypePluginParticipantData
HelloWorldPlugin_on_participant_attached(
    void *registration_data,
    const struct PRESTypePluginParticipantInfo *participant_info,
    RTIBool top_level_registration,
    void *container_plugin_context,
    RTICdrTypeCode *type_code)
{
    struct RTIXCdrInterpreterPrograms *programs = NULL;
    struct RTIXCdrInterpreterProgramsGenProperty programProperty =
    RTIXCdrInterpreterProgramsGenProperty_INITIALIZER;
    struct PRESTypePluginDefaultParticipantData *pd = NULL;
    RTIOsapiUtility_unusedParameter(registration_data);
    RTIOsapiUtility_unusedParameter(participant_info);
    RTIOsapiUtility_unusedParameter(top_level_registration);
    RTIOsapiUtility_unusedParameter(container_plugin_context);
    RTIOsapiUtility_unusedParameter(type_code);
    pd = (struct PRESTypePluginDefaultParticipantData *)
    PRESTypePluginDefaultParticipantData_new(participant_info);
    programProperty.generateV1Encapsulation = RTI_XCDR_TRUE;
    programProperty.generateV2Encapsulation = RTI_XCDR_TRUE;
    programProperty.resolveAlias = RTI_XCDR_TRUE;
    programProperty.inlineStruct = RTI_XCDR_TRUE;
    programProperty.optimizeEnum = RTI_XCDR_TRUE;
    programProperty.unboundedSize = RTIXCdrLong_MAX;
    programs = DDS_TypeCodeFactory_assert_programs_in_global_list(
        DDS_TypeCodeFactory_get_instance(),
        HelloWorld_get_typecode(),
        &programProperty,
        RTI_XCDR_PROGRAM_MASK_TYPEPLUGIN);
    if (programs == NULL) {
        PRESTypePluginDefaultParticipantData_delete(
            (PRESTypePluginParticipantData) pd);
        return NULL;
    }
    pd->programs = programs;
    return (PRESTypePluginParticipantData)pd;
}
```

```
void
HelloWorldPlugin_on_participant_detached(
    PRESTypePluginParticipantData participant_data)
{
    if (participant_data != NULL) {
        struct PRESTypePluginDefaultParticipantData *pd =
        (struct PRESTypePluginDefaultParticipantData *)participant_data;
        if (pd->programs != NULL) {
            DDS_TypeCodeFactory_remove_programs_from_global_list(
                DDS_TypeCodeFactory_get_instance(),
                pd->programs);
            pd->programs = NULL;
        }
        PRESTypePluginDefaultParticipantData_delete(participant_data);
    }
}
PRESTypePluginEndpointData
HelloWorldPlugin_on_endpoint_attached(
    PRESTypePluginParticipantData participant_data,
    const struct PRESTypePluginEndpointInfo *endpoint_info,
    RTIBool top_level_registration,
    void *containerPluginContext)
{
    PRESTypePluginEndpointData epd = NULL;
    unsigned int serializedSampleMaxSize = 0;
    RTIOsapiUtility_unusedParameter(top_level_registration);
    RTIOsapiUtility_unusedParameter(containerPluginContext);
    if (participant_data == NULL) {
        return NULL;
    }
    epd = PRESTypePluginDefaultEndpointData_new(
        participant_data,
        endpoint_info,
        HelloWorldPluginSupport_create_dataI,
        HelloWorldPluginSupport_destroy_dataI,
        NULL , NULL );
    if (epd == NULL) {
        return NULL;
    }
    if (endpoint_info->endpointKind == PRES_TYPEPLUGIN_ENDPOINT_WRITER) {
        serializedSampleMaxSize = HelloWorldPlugin_get_serialized_sample_max_size(
            epd,RTI_FALSE,RTI_CDR_ENCAPSULATION_ID_CDR_BE,0);
        PRESTypePluginDefaultEndpointData_setMaxSizeSerializedSample(epd, serializedSampleMaxSize);
        if (PRESTypePluginDefaultEndpointData_createWriterPool(
            epd,
            endpoint_info,
            (PRESTypePluginGetSerializedSampleMaxSizeFunction)
            HelloWorldPlugin_get_serialized_sample_max_size, epd,
            (PRESTypePluginGetSerializedSampleSizeFunction)
            PRESTypePlugin_interpretedGetSerializedSampleSize,
            epd) == RTI_FALSE) {
            PRESTypePluginDefaultEndpointData_delete(epd);
            return NULL;
        }
    }
    return epd;
}
void
HelloWorldPlugin_on_endpoint_detached(
    PRESTypePluginEndpointData endpoint_data)
{
    PRESTypePluginDefaultEndpointData_delete(endpoint_data);
}
void
HelloWorldPlugin_return_sample(
    PRESTypePluginEndpointData endpoint_data,
    HelloWorld *sample,
    void *handle)
{
    HelloWorld_finalize_optional_members(sample, RTI_TRUE);
    PRESTypePluginDefaultEndpointData_returnSample(
        endpoint_data, sample, handle);
}
void HelloWorldPlugin_finalize_optional_members(
    PRESTypePluginEndpointData endpoint_data,
    HelloWorld* sample,
    RTIBool deletePointers)
{
    RTIOsapiUtility_unusedParameter(endpoint_data);
    HelloWorld_finalize_optional_members(
        sample, deletePointers);
```

```
}
RTIBool
HelloWorldPlugin_copy_sample(
    PRESTypePluginEndpointData endpoint_data,
    HelloWorld *dst,
    const HelloWorld *src)
{
    RTIOsapiUtility_unusedParameter(endpoint_data);
    return HelloWorldPluginSupport_copy_data(dst,src);
}
/* ------------------------------------------------------------------------
(De)Serialize functions:
* ------------------------------------------------------------------------ */
unsigned int
HelloWorldPlugin_get_serialized_sample_max_size(
    PRESTypePluginEndpointData endpoint_data,
    RTIBool include_encapsulation,
    RTIEncapsulationId encapsulation_id,
    unsigned int current_alignment);
RTIBool
HelloWorldPlugin_serialize_to_cdr_buffer_ex(
    char *buffer,
    unsigned int *length,
    const HelloWorld *sample,
    DDS_DataRepresentationId_t representation)
{
    RTIEncapsulationId encapsulationId = RTI_CDR_ENCAPSULATION_ID_INVALID;
    struct RTICdrStream cdrStream;
    struct PRESTypePluginDefaultEndpointData epd;
    RTIBool result;
    struct PRESTypePluginDefaultParticipantData pd;
    struct RTIXCdrTypePluginProgramContext defaultProgramContext =
    RTIXCdrTypePluginProgramContext_INTIALIZER;
    struct PRESTypePlugin plugin;
    if (length == NULL) {
        return RTI_FALSE;
    }
    RTIOsapiMemory_zero(&epd, sizeof(struct PRESTypePluginDefaultEndpointData));
    epd.programContext = defaultProgramContext;
    epd._participantData = &pd;
    epd.typePlugin = &plugin;
    epd.programContext.endpointPluginData = &epd;
    plugin.typeCode = (struct RTICdrTypeCode *)
    HelloWorld_get_typecode();
    pd.programs = HelloWorldPlugin_get_programs();
    if (pd.programs == NULL) {
        return RTI_FALSE;
    }
    encapsulationId = DDS_TypeCode_get_native_encapsulation(
        (DDS_TypeCode *) plugin.typeCode,
        representation);
    if (encapsulationId == RTI_CDR_ENCAPSULATION_ID_INVALID) {
        return RTI_FALSE;
    }
    epd._maxSizeSerializedSample =
    HelloWorldPlugin_get_serialized_sample_max_size(
        (PRESTypePluginEndpointData)&epd,
        RTI_TRUE,
        encapsulationId,
        0);
    if (buffer == NULL) {
        *length =
        PRESTypePlugin_interpretedGetSerializedSampleSize(
            (PRESTypePluginEndpointData)&epd,
            RTI_TRUE,
            encapsulationId,
            0,
            sample);
        if (*length == 0) {
            return RTI_FALSE;
        }
        return RTI_TRUE;
    }
    RTICdrStream_init(&cdrStream);
    RTICdrStream_set(&cdrStream, (char *)buffer, *length);
    result = PRESTypePlugin_interpretedSerialize(
        (PRESTypePluginEndpointData)&epd,
        sample,
        &cdrStream,
        RTI_TRUE,
        encapsulationId,
```

```
        RTI_TRUE,
        NULL);
    *length = (unsigned int) RTICdrStream_getCurrentPositionOffset(&cdrStream);
    return result;
}
RTIBool
HelloWorldPlugin_serialize_to_cdr_buffer(
    char *buffer,
    unsigned int *length,
    const HelloWorld *sample)
{
    return HelloWorldPlugin_serialize_to_cdr_buffer_ex(
        buffer,
        length,
        sample,
        DDS_AUTO_DATA_REPRESENTATION);
}
RTIBool
HelloWorldPlugin_deserialize_from_cdr_buffer(
    HelloWorld *sample,
    const char * buffer,
    unsigned int length)
{
    struct RTICdrStream cdrStream;
    struct PRESTypePluginDefaultEndpointData epd;
    struct RTIXCdrTypePluginProgramContext defaultProgramContext =
    RTIXCdrTypePluginProgramContext_INTIALIZER;
    struct PRESTypePluginDefaultParticipantData pd;
    struct PRESTypePlugin plugin;
    epd.programContext = defaultProgramContext;
    epd._participantData = &pd;
    epd.typePlugin = &plugin;
    epd.programContext.endpointPluginData = &epd;
    plugin.typeCode = (struct RTICdrTypeCode *)
    HelloWorld_get_typecode();
    pd.programs = HelloWorldPlugin_get_programs();
    if (pd.programs == NULL) {
        return RTI_FALSE;
    }
    RTIXCdrSampleAssignabilityProperty_setFromGlobalComplianceMask(
        &epd._assignabilityProperty);
    RTICdrStream_init(&cdrStream);
    RTICdrStream_set(&cdrStream, (char *)buffer, length);
    HelloWorld_finalize_optional_members(sample, RTI_TRUE);
    return PRESTypePlugin_interpretedDeserialize(
        (PRESTypePluginEndpointData)&epd, sample,
        &cdrStream, RTI_TRUE, RTI_TRUE,
        NULL);
}
#if !defined(NDDS_STANDALONE_TYPE)
DDS_ReturnCode_t
HelloWorldPlugin_data_to_string(
    const HelloWorld *sample,
    char *_str,
    DDS_UnsignedLong *str_size,
    const struct DDS_PrintFormatProperty *property)
{
    DDS_DynamicData *data = NULL;
    char *buffer = NULL;
    unsigned int length = 0;
    struct DDS_PrintFormat printFormat;
    DDS_ReturnCode_t retCode = DDS_RETCODE_ERROR;
    if (sample == NULL) {
        return DDS_RETCODE_BAD_PARAMETER;
    }
    if (str_size == NULL) {
        return DDS_RETCODE_BAD_PARAMETER;
    }
    if (property == NULL) {
        return DDS_RETCODE_BAD_PARAMETER;
    }
    if (!HelloWorldPlugin_serialize_to_cdr_buffer(
        NULL,
        &length,
        sample)) {
        return DDS_RETCODE_ERROR;
    }
    RTIOsapiHeap_allocateBuffer(&buffer, length, RTI_OSAPI_ALIGNMENT_DEFAULT);
    if (buffer == NULL) {
        return DDS_RETCODE_ERROR;
    }
```

```
    if (!HelloWorldPlugin_serialize_to_cdr_buffer(
        buffer,
        &length,
        sample)) {
        RTIOsapiHeap_freeBuffer(buffer);
        return DDS_RETCODE_ERROR;
    }
    data = DDS_DynamicData_new(
        HelloWorld_get_typecode(),
        &DDS_DYNAMIC_DATA_PROPERTY_DEFAULT);
    if (data == NULL) {
        RTIOsapiHeap_freeBuffer(buffer);
        return DDS_RETCODE_ERROR;
    }
    retCode = DDS_DynamicData_from_cdr_buffer(data, buffer, length);
    if (retCode != DDS_RETCODE_OK) {
        RTIOsapiHeap_freeBuffer(buffer);
        DDS_DynamicData_delete(data);
        return retCode;
    }
    retCode = DDS_PrintFormatProperty_to_print_format(
        property,
        &printFormat);
    if (retCode != DDS_RETCODE_OK) {
        RTIOsapiHeap_freeBuffer(buffer);
        DDS_DynamicData_delete(data);
        return retCode;
    }
    retCode = DDS_DynamicDataFormatter_to_string_w_format(
        data,
        _str,
        str_size,
        &printFormat);
    if (retCode != DDS_RETCODE_OK) {
        RTIOsapiHeap_freeBuffer(buffer);
        DDS_DynamicData_delete(data);
        return retCode;
    }
    RTIOsapiHeap_freeBuffer(buffer);
    DDS_DynamicData_delete(data);
    return DDS_RETCODE_OK;
}
#endif
unsigned int
HelloWorldPlugin_get_serialized_sample_max_size(
    PRESTypePluginEndpointData endpoint_data,
    RTIBool include_encapsulation,
    RTIEncapsulationId encapsulation_id,
    unsigned int current_alignment)
{
    unsigned int size;
    RTIBool overflow = RTI_FALSE;
    size = PRESTypePlugin_interpretedGetSerializedSampleMaxSize(
        endpoint_data,&overflow,include_encapsulation,encapsulation_id,current_alignment);
    if (overflow) {
        size = RTI_CDR_MAX_SERIALIZED_SIZE;
    }
    return size;
}
/* ------------------------------------------------------------------------------------------
Key Management functions:
* ------------------------------------------------------------------------------------------ */
PRESTypePluginKeyKind
HelloWorldPlugin_get_key_kind(void)
{
    return PRES_TYPEPLUGIN_NO_KEY;
}
RTIBool HelloWorldPlugin_deserialize_key(
    PRESTypePluginEndpointData endpoint_data,
    HelloWorld **sample,
    RTIBool * drop_sample,
    struct RTICdrStream *cdrStream,
    RTIBool deserialize_encapsulation,
    RTIBool deserialize_key,
    void *endpoint_plugin_qos)
{
    RTIBool result;
    RTIOsapiUtility_unusedParameter(drop_sample);
    /*  Depending on the type and the flags used in rtiddsgen, coverity may detect
    that sample is always null. Since the case is very dependant on
    the IDL/XML and the configuration we keep the check for safety.
```

```
    */
    result= PRESTypePlugin_interpretedDeserializeKey(
        endpoint_data,
        /* coverity[check_after_deref] */
        (sample != NULL) ? *sample : NULL,
        cdrStream,
        deserialize_encapsulation,
        deserialize_key,
        endpoint_plugin_qos);
    return result;
}
unsigned int
HelloWorldPlugin_get_serialized_key_max_size(
    PRESTypePluginEndpointData endpoint_data,
    RTIBool include_encapsulation,
    RTIEncapsulationId encapsulation_id,
    unsigned int current_alignment)
{
    unsigned int size;
    RTIBool overflow = RTI_FALSE;
    size = PRESTypePlugin_interpretedGetSerializedKeyMaxSize(
        endpoint_data,&overflow,include_encapsulation,encapsulation_id,current_alignment);
    if (overflow) {
        size = RTI_CDR_MAX_SERIALIZED_SIZE;
    }
    return size;
}
unsigned int
HelloWorldPlugin_get_serialized_key_max_size_for_keyhash(
    PRESTypePluginEndpointData endpoint_data,
    RTIEncapsulationId encapsulation_id,
    unsigned int current_alignment)
{
    unsigned int size;
    RTIBool overflow = RTI_FALSE;
    size = PRESTypePlugin_interpretedGetSerializedKeyMaxSizeForKeyhash(
        endpoint_data,
        &overflow,
        encapsulation_id,
        current_alignment);
    if (overflow) {
        size = RTI_CDR_MAX_SERIALIZED_SIZE;
    }
    return size;
}
struct RTIXCdrInterpreterPrograms * HelloWorldPlugin_get_programs(void)
{
    return ::rti::xcdr::get_cdr_serialization_programs<
    HelloWorld,
    true, true, true>();
}
/* ------------------------------------------------------------------------
* Plug-in Installation Methods
* ------------------------------------------------------------------------ */
struct PRESTypePlugin *HelloWorldPlugin_new(void)
{
    struct PRESTypePlugin *plugin = NULL;
    const struct PRESTypePluginVersion PLUGIN_VERSION =
    PRES_TYPE_PLUGIN_VERSION_2_0;
    RTIOsapiHeap_allocateStructure(
        &plugin, struct PRESTypePlugin);
    if (plugin == NULL) {
        return NULL;
    }
    plugin->version = PLUGIN_VERSION;
    /* set up parent's function pointers */
    plugin->onParticipantAttached =
    (PRESTypePluginOnParticipantAttachedCallback)
    HelloWorldPlugin_on_participant_attached;
    plugin->onParticipantDetached =
    (PRESTypePluginOnParticipantDetachedCallback)
    HelloWorldPlugin_on_participant_detached;
    plugin->onEndpointAttached =
    (PRESTypePluginOnEndpointAttachedCallback)
    HelloWorldPlugin_on_endpoint_attached;
    plugin->onEndpointDetached =
    (PRESTypePluginOnEndpointDetachedCallback)
    HelloWorldPlugin_on_endpoint_detached;
    plugin->copySampleFnc =
    (PRESTypePluginCopySampleFunction)
    HelloWorldPlugin_copy_sample;
```

```
    plugin->createSampleFnc =
    (PRESTypePluginCreateSampleFunction)
    HelloWorldPlugin_create_sample;
    plugin->destroySampleFnc =
    (PRESTypePluginDestroySampleFunction)
    HelloWorldPlugin_destroy_sample;
    plugin->finalizeOptionalMembersFnc =
    (PRESTypePluginFinalizeOptionalMembersFunction)
    HelloWorldPlugin_finalize_optional_members;
    plugin->serializeFnc =
    (PRESTypePluginSerializeFunction) PRESTypePlugin_interpretedSerialize;
    plugin->deserializeFnc =
    (PRESTypePluginDeserializeFunction) PRESTypePlugin_interpretedDeserializeWithAlloc;
    plugin->getSerializedSampleMaxSizeFnc =
    (PRESTypePluginGetSerializedSampleMaxSizeFunction)
    HelloWorldPlugin_get_serialized_sample_max_size;
    plugin->getSerializedSampleMinSizeFnc =
    (PRESTypePluginGetSerializedSampleMinSizeFunction)
    PRESTypePlugin_interpretedGetSerializedSampleMinSize;
    plugin->getDeserializedSampleMaxSizeFnc = NULL;
    plugin->getSampleFnc =
    (PRESTypePluginGetSampleFunction)
    HelloWorldPlugin_get_sample;
    plugin->returnSampleFnc =
    (PRESTypePluginReturnSampleFunction)
    HelloWorldPlugin_return_sample;
    plugin->getKeyKindFnc =
    (PRESTypePluginGetKeyKindFunction)
    HelloWorldPlugin_get_key_kind;
    /* These functions are only used for keyed types. As this is not a keyed
    type they are all set to NULL
    */
    plugin->serializeKeyFnc = NULL ;
    plugin->deserializeKeyFnc = NULL;
    plugin->getKeyFnc = NULL;
    plugin->returnKeyFnc = NULL;
    plugin->instanceToKeyFnc = NULL;
    plugin->keyToInstanceFnc = NULL;
    plugin->getSerializedKeyMaxSizeFnc = NULL;
    plugin->instanceToKeyHashFnc = NULL;
    plugin->serializedSampleToKeyHashFnc = NULL;
    plugin->serializedKeyToKeyHashFnc = NULL;
    #ifdef NDDS_STANDALONE_TYPE
    plugin->typeCode = NULL;
    #else
    plugin->typeCode =  (struct RTICdrTypeCode *)HelloWorld_get_typecode();
    #endif
    plugin->languageKind = PRES_TYPEPLUGIN_CPP_LANG;
    /* Serialized buffer */
    plugin->getBuffer =
    (PRESTypePluginGetBufferFunction)
    HelloWorldPlugin_get_buffer;
    plugin->returnBuffer =
    (PRESTypePluginReturnBufferFunction)
    HelloWorldPlugin_return_buffer;
    plugin->getBufferWithParams = NULL;
    plugin->returnBufferWithParams = NULL;
    plugin->getSerializedSampleSizeFnc =
    (PRESTypePluginGetSerializedSampleSizeFunction)
    PRESTypePlugin_interpretedGetSerializedSampleSize;
    plugin->getWriterLoanedSampleFnc = NULL;
    plugin->returnWriterLoanedSampleFnc = NULL;
    plugin->returnWriterLoanedSampleFromCookieFnc = NULL;
    plugin->validateWriterLoanedSampleFnc = NULL;
    plugin->setWriterLoanedSampleSerializedStateFnc = NULL;
    plugin->endpointTypeName = HelloWorldTYPENAME;
    plugin->isMetpType = RTI_FALSE;
    plugin->isRecursiveType = RTI_FALSE;
    return plugin;
}
void
HelloWorldPlugin_delete(struct PRESTypePlugin *plugin)
{
    RTIOsapiHeap_freeStructure(plugin);
}
#undef RTI_CDR_CURRENT_SUBMODULE
```

11.5 HelloWorld_publisher.cxx

11.5.1 RTI Connext Publication Example

The publication example generated by rtiddsgen (see the `Code Generator User's Manual` for more information). The example has been modified slightly to update the sample value.

11.5.1.1 HelloWorld_publisher.cxx

```
/*
* (c) Copyright, Real-Time Innovations, 2020.  All rights reserved.
* RTI grants Licensee a license to use, modify, compile, and create derivative
* works of the software solely for use with RTI Connext DDS. Licensee may
* redistribute copies of the software provided that all such copies are subject
* to this license. The software is provided "as is", with no warranty of any
* type, including any warranty for fitness for any purpose. RTI is under no
* obligation to maintain or support the software. RTI shall not be liable for
* any incidental or consequential damages arising out of the use or inability
* to use the software.
*/
#include <iostream>
#include <stdio.h>
#include <stdlib.h>
#include "HelloWorld.h"
#include "HelloWorldSupport.h"
#include "ndds/ndds_cpp.h"
#include "application.h"
using namespace application;
static int shutdown_participant(
    DDSDomainParticipant *participant,
    const char *shutdown_message,
    int status);
int run_publisher_application(unsigned int domain_id, unsigned int sample_count)
{
    // Start communicating in a domain, usually one participant per application
    DDSDomainParticipant *participant =
    DDSTheParticipantFactory->create_participant_with_profile(
        domain_id,
        "HelloWorld_Library",
        "Participant_Profile",
        NULL /* listener */,
        DDS_STATUS_MASK_NONE);
    if (participant == NULL) {
        return shutdown_participant(participant, "create_participant error", EXIT_FAILURE);
    }
    // A Publisher allows an application to create one or more DataWriters
    DDSPublisher *publisher = participant->create_publisher(
        DDS_PUBLISHER_QOS_DEFAULT,
        NULL /* listener */,
        DDS_STATUS_MASK_NONE);
    if (publisher == NULL) {
        return shutdown_participant(participant, "create_publisher error", EXIT_FAILURE);
    }
    // Register the datatype to use when creating the Topic
    const char *type_name = HelloWorldTypeSupport::get_type_name();
    DDS_ReturnCode_t retcode =
    HelloWorldTypeSupport::register_type(participant, type_name);
    if (retcode != DDS_RETCODE_OK) {
        return shutdown_participant(participant, "register_type error", EXIT_FAILURE);
    }
    // Create a Topic with a name and a datatype
    DDSTopic *topic = participant->create_topic(
        "Example HelloWorld",
        type_name,
        DDS_TOPIC_QOS_DEFAULT,
        NULL /* listener */,
        DDS_STATUS_MASK_NONE);
    if (topic == NULL) {
        return shutdown_participant(participant, "create_topic error", EXIT_FAILURE);
    }
    // This DataWriter writes data on "Example HelloWorld" Topic
    DDSDataWriter *untyped_writer = publisher->create_datawriter_with_profile(
        topic,
```

```
        "HelloWorld_Library", // QoS Library name
        "Periodic_Best_Effort_Profile",  // QoS Profile name
        NULL /* listener */,
        DDS_STATUS_MASK_NONE);
    if (untyped_writer == NULL) {
        return shutdown_participant(participant, "create_datawriter error", EXIT_FAILURE);
    }
    // Narrow casts from an untyped DataWriter to a writer of your type
    HelloWorldDataWriter *typed_writer =
    HelloWorldDataWriter::narrow(untyped_writer);
    if (typed_writer == NULL) {
        return shutdown_participant(participant, "DataWriter narrow error", EXIT_FAILURE);
    }
    // Create data for writing, allocating all members
    HelloWorld *data = HelloWorldTypeSupport::create_data();
    if (data == NULL) {
        return shutdown_participant(
            participant,
            "HelloWorldTypeSupport::create_data error",
            EXIT_FAILURE);
    }
    // Main loop, write data
    for (unsigned int samples_written = 0;
    !shutdown_requested && samples_written < sample_count;
    ++samples_written) {
        // Modify the data to be written here
        std::cout << "Writing HelloWorld, count " << samples_written
        << std::endl;
        retcode = typed_writer->write(*data, DDS_HANDLE_NIL);
        if (retcode != DDS_RETCODE_OK) {
            std::cerr << "write error " << retcode << std::endl;
        }
        // Send once every second
        DDS_Duration_t send_period = { 1, 0 };
        NDDSUtility::sleep(send_period);
    }
    // Delete previously allocated HelloWorld, including all contained elements
    retcode = HelloWorldTypeSupport::delete_data(data);
    if (retcode != DDS_RETCODE_OK) {
        std::cerr << "HelloWorldTypeSupport::delete_data error " << retcode
        << std::endl;
    }
    // Delete all entities (DataWriter, Topic, Publisher, DomainParticipant)
    return shutdown_participant(participant, "Shutting down", EXIT_SUCCESS);
}
// Delete all entities
static int shutdown_participant(
    DDSDomainParticipant *participant,
    const char *shutdown_message,
    int status)
{
    DDS_ReturnCode_t retcode;
    std::cout << shutdown_message << std::endl;
    if (participant != NULL) {
        // Cleanup everything created by this Participant
        retcode = participant->delete_contained_entities();
        if (retcode != DDS_RETCODE_OK) {
            std::cerr << "delete_contained_entities error " << retcode
            << std::endl;
            status = EXIT_FAILURE;
        }
        retcode = DDSTheParticipantFactory->delete_participant(participant);
        if (retcode != DDS_RETCODE_OK) {
            std::cerr << "delete_participant error " << retcode << std::endl;
            status = EXIT_FAILURE;
        }
    }
    return status;
}
int main(int argc, char *argv[])
{
    // Parse arguments and handle control-C
    ApplicationArguments arguments;
    parse_arguments(arguments, argc, argv);
    if (arguments.parse_result == PARSE_RETURN_EXIT) {
        return EXIT_SUCCESS;
    } else if (arguments.parse_result == PARSE_RETURN_FAILURE) {
        return EXIT_FAILURE;
    }
    setup_signal_handlers();
    // Sets Connext verbosity to help debugging
```

```
    NDDSConfigLogger::get_instance()->set_verbosity(arguments.verbosity);
    int status = run_publisher_application(arguments.domain_id, arguments.sample_count);
    // Releases the memory used by the participant factory.  Optional at
    // application exit
    DDS_ReturnCode_t retcode = DDSDomainParticipantFactory::finalize_instance();
    if (retcode != DDS_RETCODE_OK) {
        std::cerr « "finalize_instance error " « retcode « std::endl;
        status = EXIT_FAILURE;
    }
    return status;
}
```

11.6 HelloWorld_subscriber.cxx

11.6.1 RTI Connext Subscription Example

The unmodified subscription example generated by rtiddsgen (see the `Code Generator User's Manual` for more information).

11.6.1.1 HelloWorld_subscriber.cxx

```
/*
* (c) Copyright, Real-Time Innovations, 2020.  All rights reserved.
* RTI grants Licensee a license to use, modify, compile, and create derivative
* works of the software solely for use with RTI Connext DDS. Licensee may
* redistribute copies of the software provided that all such copies are subject
* to this license. The software is provided "as is", with no warranty of any
* type, including any warranty for fitness for any purpose. RTI is under no
* obligation to maintain or support the software. RTI shall not be liable for
* any incidental or consequential damages arising out of the use or inability
* to use the software.
*/
#include <iostream>
#include <stdio.h>
#include <stdlib.h>
#include "HelloWorld.h"
#include "HelloWorldSupport.h"
#include "ndds/ndds_cpp.h"
#include "application.h"
using namespace application;
static int shutdown_participant(
    DDSDomainParticipant *participant,
    const char *shutdown_message,
    int status);
// Process data. Returns number of samples processed.
unsigned int process_data(HelloWorldDataReader *typed_reader)
{
    HelloWorldSeq data_seq;     // Sequence of received data
    DDS_SampleInfoSeq info_seq; // Metadata associated with samples in data_seq
    unsigned int samples_read = 0;
    // Take available data from DataReader's queue
    typed_reader->take(
        data_seq,
        info_seq,
        DDS_LENGTH_UNLIMITED,
        DDS_ANY_SAMPLE_STATE,
        DDS_ANY_VIEW_STATE,
        DDS_ANY_INSTANCE_STATE);
    // Iterate over all available data
    for (int i = 0; i < data_seq.length(); ++i) {
        // Check if a sample is an instance lifecycle event
        if (info_seq[i].valid_data) {
            // Print data
            std::cout « "Received data" « std::endl;
            HelloWorldTypeSupport::print_data(&data_seq[i]);
            samples_read++;
        } else {  // This is an instance lifecycle event with no data payload.
            std::cout « "Received instance state notification" « std::endl;
        }
    }
```

```
    // Data loaned from Connext for performance. Return loan when done.
    DDS_ReturnCode_t retcode = typed_reader->return_loan(data_seq, info_seq);
    if (retcode != DDS_RETCODE_OK) {
        std::cerr « "return loan error " « retcode « std::endl;
    }
    return samples_read;
}
int run_subscriber_application(unsigned int domain_id, unsigned int sample_count)
{
    // Start communicating in a domain, usually one participant per application
    DDSDomainParticipant *participant =
    DDSTheParticipantFactory->create_participant_with_profile(
        domain_id,
        "HelloWorld_Library",
        "Participant_Profile",
        NULL /* listener */,
        DDS_STATUS_MASK_NONE);
    if (participant == NULL) {
        return shutdown_participant(participant, "create_participant error", EXIT_FAILURE);
    }
    // A Subscriber allows an application to create one or more DataReaders
    DDSSubscriber *subscriber = participant->create_subscriber(
        DDS_SUBSCRIBER_QOS_DEFAULT,
        NULL /* listener */,
        DDS_STATUS_MASK_NONE);
    if (subscriber == NULL) {
        return shutdown_participant(participant, "create_subscriber error", EXIT_FAILURE);
    }
    // Register the datatype to use when creating the Topic
    const char *type_name = HelloWorldTypeSupport::get_type_name();
    DDS_ReturnCode_t retcode =
    HelloWorldTypeSupport::register_type(participant, type_name);
    if (retcode != DDS_RETCODE_OK) {
        return shutdown_participant(participant, "register_type error", EXIT_FAILURE);
    }
    // Create a Topic with a name and a datatype
    DDSTopic *topic = participant->create_topic(
        "Example HelloWorld",
        type_name,
        DDS_TOPIC_QOS_DEFAULT,
        NULL /* listener */,
        DDS_STATUS_MASK_NONE);
    if (topic == NULL) {
        return shutdown_participant(participant, "create_topic error", EXIT_FAILURE);
    }
    // This DataReader reads data on "Example HelloWorld" Topic
    DDSDataReader *untyped_reader = subscriber->create_datareader_with_profile(
        topic,
        "HelloWorld_Library", // QoS Library name
        "Periodic_Best_Effort_Profile",  // QoS Profile name
        NULL,
        DDS_STATUS_MASK_NONE);
    if (untyped_reader == NULL) {
        return shutdown_participant(participant, "create_datareader error", EXIT_FAILURE);
    }
    // Narrow casts from a untyped DataReader to a reader of your type
    HelloWorldDataReader *typed_reader =
    HelloWorldDataReader::narrow(untyped_reader);
    if (typed_reader == NULL) {
        return shutdown_participant(participant, "DataReader narrow error", EXIT_FAILURE);
    }
    // Create ReadCondition that triggers when unread data in reader's queue
    DDSReadCondition *read_condition = typed_reader->create_readcondition(
        DDS_NOT_READ_SAMPLE_STATE,
        DDS_ANY_VIEW_STATE,
        DDS_ANY_INSTANCE_STATE);
    if (read_condition == NULL) {
        return shutdown_participant(participant, "create_readcondition error", EXIT_FAILURE);
    }
    // WaitSet will be woken when the attached condition is triggered
    DDSWaitSet waitset;
    retcode = waitset.attach_condition(read_condition);
    if (retcode != DDS_RETCODE_OK) {
        return shutdown_participant(participant, "attach_condition error", EXIT_FAILURE);
    }
    // Main loop. Wait for data to arrive, and process when it arrives
    unsigned int samples_read = 0;
    while (!shutdown_requested && samples_read < sample_count) {
        DDSConditionSeq active_conditions_seq;
        // Wait for data and report if it does not arrive in 1 second
        DDS_Duration_t wait_timeout = { 1, 0 };
```

```
        retcode = waitset.wait(active_conditions_seq, wait_timeout);
        if (retcode == DDS_RETCODE_OK) {
            // If the read condition is triggered, process data
            samples_read += process_data(typed_reader);
        } else {
            if (retcode == DDS_RETCODE_TIMEOUT) {
                std::cout << "No data after 1 second" << std::endl;
            }
        }
    }
    // Cleanup
    return shutdown_participant(participant, "Shutting down", 0);
}
// Delete all entities
static int shutdown_participant(
    DDSDomainParticipant *participant,
    const char *shutdown_message,
    int status)
{
    DDS_ReturnCode_t retcode;
    std::cout << shutdown_message << std::endl;
    if (participant != NULL) {
        // Cleanup everything created by this Participant
        retcode = participant->delete_contained_entities();
        if (retcode != DDS_RETCODE_OK) {
            std::cerr << "delete_contained_entities error" << retcode
            << std::endl;
            status = EXIT_FAILURE;
        }
        retcode = DDSTheParticipantFactory->delete_participant(participant);
        if (retcode != DDS_RETCODE_OK) {
            std::cerr << "delete_participant error" << retcode << std::endl;
            status = EXIT_FAILURE;
        }
    }
    return status;
}
int main(int argc, char *argv[])
{
    // Parse arguments and handle control-C
    ApplicationArguments arguments;
    parse_arguments(arguments, argc, argv);
    if (arguments.parse_result == PARSE_RETURN_EXIT) {
        return EXIT_SUCCESS;
    } else if (arguments.parse_result == PARSE_RETURN_FAILURE) {
        return EXIT_FAILURE;
    }
    setup_signal_handlers();
    // Sets Connext verbosity to help debugging
    NDDSConfigLogger::get_instance()->set_verbosity(arguments.verbosity);
    int status = run_subscriber_application(arguments.domain_id, arguments.sample_count);
    // Releases the memory used by the participant factory.  Optional at
    // application exit
    DDS_ReturnCode_t retcode = DDSDomainParticipantFactory::finalize_instance();
    if (retcode != DDS_RETCODE_OK) {
        std::cerr << "finalize_instance error" << retcode << std::endl;
        status = EXIT_FAILURE;
    }
    return status;
}
```

Index

_d
    MyFlatUnionOffset, 2036
~AbstractBuilder
    rti::flat::AbstractBuilder, 787
~DDSAsyncWaitSet
    DDSAsyncWaitSet, 1533
~DDSDataReaderStatusConditionHandler
    DDSDataReaderStatusConditionHandler, 1587
~DDSDynamicDataTypeSupport
    DDSDynamicDataTypeSupport, 1726
~DDSGuardCondition
    DDSGuardCondition, 1740
~DDSWaitSet
    DDSWaitSet, 1900
~DDS_DynamicData
    DDS_DynamicData, 1030
~DDS_KeyedOctets
    DDS_KeyedOctets, 1170
~DDS_KeyedString
    DDS_KeyedString, 1172
~DDS_Octets
    DDS_Octets, 1218
~FooSeq
    FooSeq, 1966
~LoanedSamples
    connext::LoanedSamples< T >, 1999
~Replier
    connext::Replier< TReq, TRep >, 2144
~Requester
    connext::Requester< TReq, TRep >, 2162
~SimpleReplier
    connext::SimpleReplier< TReq, TRep >, 2207

absolute_generation_rank
    DDS_SampleInfo, 1350
accept_unknown_peers
    DDS_DiscoveryQosPolicy, 963
access
    DDS_ValueMember, 1507
access_scope
    DDS_PresentationQosPolicy, 1252
acknowledge_all
    DDSDataReader, 1565, 1566
acknowledge_sample
    DDSDataReader, 1564, 1565
acknowledgment_kind
    DDS_ReliabilityQosPolicy, 1298
active_count
    DDS_ReliableReaderActivityChangedStatus, 1301
active_count_change
    DDS_ReliableReaderActivityChangedStatus, 1301
Activity Context, 729
    NDDS_CONFIG_ACTIVITY_CONTEXT_ATTRIBUTE_DOMAIN_ID, 733
    NDDS_CONFIG_ACTIVITY_CONTEXT_ATTRIBUTE_ENTITY_KIND, 733
    NDDS_CONFIG_ACTIVITY_CONTEXT_ATTRIBUTE_ENTITY_NAME, 733
    NDDS_CONFIG_ACTIVITY_CONTEXT_ATTRIBUTE_GUID_PREFIX, 732
    NDDS_CONFIG_ACTIVITY_CONTEXT_ATTRIBUTE_MASK_ALL, 731
    NDDS_CONFIG_ACTIVITY_CONTEXT_ATTRIBUTE_MASK_DEFAUL 730
    NDDS_CONFIG_ACTIVITY_CONTEXT_ATTRIBUTE_MASK_NONE, 731
    NDDS_CONFIG_ACTIVITY_CONTEXT_ATTRIBUTE_TOPIC, 732
    NDDS_CONFIG_ACTIVITY_CONTEXT_ATTRIBUTE_TYPE, 732
    NDDS_Config_ActivityContextAttributeKind, 731
    NDDS_Config_ActivityContextAttributeKindMask, 731
    set_attribute_mask, 733
add_member
    DDS_TypeCode, 1463
add_member_ex
    DDS_TypeCode, 1465
add_member_to_enum
    DDS_TypeCode, 1461
add_member_to_union
    DDS_TypeCode, 1462
add_my_final
    MyFlatMutableBuilder, 2023
    MyFlatUnionBuilder, 2034
add_my_final_array
    MyFlatMutableBuilder, 2023
add_my_optional_primitive
    MyFlatMutableBuilder, 2022
add_my_primitive
    MyFlatMutableBuilder, 2022

MyFlatUnionBuilder, 2034
add_my_primitive_array
MyFlatMutableBuilder, 2022
add_n
rti::flat::FinalSequenceBuilder< ElementOffset >, 1915
rti::flat::PrimitiveSequenceBuilder< T >, 2136, 2137
add_next
rti::flat::FinalSequenceBuilder< ElementOffset >, 1914
rti::flat::PrimitiveSequenceBuilder< T >, 2136
add_peer
DDSDomainParticipant, 1678
add_pointer_property
PROPERTY, 624
add_property
PROPERTY, 622
add_receive_route
NDDSTransportSupport, 2107
add_send_route
NDDSTransportSupport, 2106
add_tag
DATA_TAG, 577
Additional RTI Connext Communication Patterns, 438
address
DDS_Locator_t, 1184
NDDS_Transport_Interface_t, 2043
address_bit_count
NDDS_Transport_Property_t, 2046
addresses
DDS_TransportMulticastMapping_t, 1411
advance
rti::flat::SequenceIterator< E, OffsetKind >, 2202
AggregationBuilders.hpp, 2281
alert
NDDSConfigLogger, 2098
alive_count
DDS_LivelinessChangedStatus, 1178
alive_count_change
DDS_LivelinessChangedStatus, 1178
alive_instance_count
DDS_DataReaderCacheStatus, 840
DDS_DataWriterCacheStatus, 891
alive_instance_count_peak
DDS_DataReaderCacheStatus, 840
DDS_DataWriterCacheStatus, 891
alive_instance_removal
DDS_DataReaderResourceLimitsInstanceReplacementSettings, 867
All QoS Policies, 549
DDS_ASYNCHRONOUSPUBLISHER_QOS_POLICY_ID, 559
DDS_AVAILABILITY_QOS_POLICY_ID, 559
DDS_BATCH_QOS_POLICY_ID, 559
DDS_DATA_REPRESENTATION_QOS_POLICY_ID, 558
DDS_DATABASE_QOS_POLICY_ID, 559
DDS_DATAREADERPROTOCOL_QOS_POLICY_ID, 559
DDS_DATAREADERRESOURCELIMITS_QOS_POLICY_ID, 558
DDS_DATATAG_QOS_POLICY_ID, 558
DDS_DATAWRITERPROTOCOL_QOS_POLICY_ID, 559
DDS_DATAWRITERRESOURCELIMITS_QOS_POLICY_ID, 558
DDS_DATAWRITERTRANSFERMODE_QOS_POLICY_ID, 560
DDS_DEADLINE_QOS_POLICY_ID, 557
DDS_DESTINATIONORDER_QOS_POLICY_ID, 558
DDS_DISCOVERY_QOS_POLICY_ID, 558
DDS_DISCOVERYCONFIG_QOS_POLICY_ID, 559
DDS_DOMAINPARTICIPANTRESOURCELIMITS_QOS_POLICY_ID, 559
DDS_DURABILITY_QOS_POLICY_ID, 557
DDS_DURABILITYSERVICE_QOS_POLICY_ID, 558
DDS_ENTITYFACTORY_QOS_POLICY_ID, 558
DDS_ENTITYNAME_QOS_POLICY_ID, 559
DDS_EVENT_QOS_POLICY_ID, 559
DDS_GROUPDATA_QOS_POLICY_ID, 558
DDS_HISTORY_QOS_POLICY_ID, 558
DDS_INVALID_QOS_POLICY_ID, 557
DDS_LATENCYBUDGET_QOS_POLICY_ID, 557
DDS_LIFESPAN_QOS_POLICY_ID, 558
DDS_LIVELINESS_QOS_POLICY_ID, 558
DDS_LOCATORFILTER_QOS_POLICY_ID, 559
DDS_LOGGING_QOS_POLICY_ID, 560
DDS_MONITORING_QOS_POLICY_ID, 560
DDS_MULTICHANNEL_QOS_POLICY_ID, 559
DDS_OWNERSHIP_QOS_POLICY_ID, 557
DDS_OWNERSHIPSTRENGTH_QOS_POLICY_ID, 558
DDS_PARTITION_QOS_POLICY_ID, 558
DDS_PRESENTATION_QOS_POLICY_ID, 557
DDS_PROFILE_QOS_POLICY_ID, 559
DDS_PROPERTY_QOS_POLICY_ID, 559
DDS_PUBLISHMODE_QOS_POLICY_ID, 559
DDS_QOS_POLICY_COUNT, 557
DDS_QosPolicyId_t, 557
DDS_QosPrintFormat_INITIALIZER, 556
DDS_READERDATALIFECYCLE_QOS_POLICY_ID, 558
DDS_RECEIVERPOOL_QOS_POLICY_ID, 559
DDS_RELIABILITY_QOS_POLICY_ID, 558
DDS_RESOURCELIMITS_QOS_POLICY_ID, 558
DDS_SYSTEMRESOURCELIMITS_QOS_POLICY_ID,

559
DDS_TIMEBASEDFILTER_QOS_POLICY_ID, 558
DDS_TOPICDATA_QOS_POLICY_ID, 558
DDS_TOPICQUERYDISPATCH_QOS_POLICY_ID, 560
DDS_TRANSPORTBUILTIN_QOS_POLICY_ID, 559
DDS_TRANSPORTMULTICAST_QOS_POLICY_ID, 559
DDS_TRANSPORTMULTICASTMAPPING_QOS_POLICY_ID, 559
DDS_TRANSPORTPRIORITY_QOS_POLICY_ID, 558
DDS_TRANSPORTSELECTION_QOS_POLICY_ID, 559
DDS_TRANSPORTUNICAST_QOS_POLICY_ID, 559
DDS_TYPE_CONSISTENCY_ENFORCEMENT_QOS_POLICY_ID, 558
DDS_TYPESUPPORT_QOS_POLICY_ID, 559
DDS_USERDATA_QOS_POLICY_ID, 557
DDS_WIREPROTOCOL_QOS_POLICY_ID, 558
DDS_WRITERDATALIFECYCLE_QOS_POLICY_ID, 558
allocate_optional_members
DDS_TypeAllocationParams_t, 1428
allocate_pointers
DDS_TypeAllocationParams_t, 1428
allow_interfaces_list
NDDS_Transport_Property_t, 2047
allow_interfaces_list_length
NDDS_Transport_Property_t, 2048
allow_multicast_interfaces_list
NDDS_Transport_Property_t, 2049
allow_multicast_interfaces_list_length
NDDS_Transport_Property_t, 2050
app_ack_period
DDS_RtpsReliableReaderProtocol_t, 1318
append_to_expression_parameter
DDSContentFilteredTopic, 1554
Application Acknowledgement, 253
Application Startup, 256
Application-Defined Metadata, 254
application_name
DDS_MonitoringQosPolicy, 1211
Applied Per-Instance, 375
array_dimension
DDS_TypeCode, 1450
array_dimension_count
DDS_TypeCode, 1449
as_readconditionparams
DDS_QueryConditionParams, 1288
assert_liveliness
DDSDataWriter, 1597
DDSDomainParticipant, 1667
assert_pointer_property
PROPERTY, 623
assert_property
PROPERTY, 622
assert_tag
DATA_TAG, 576
assertions_per_lease_duration
DDS_LivelinessQosPolicy, 1183
asynchronous_batch_thread
DDS_AsynchronousPublisherQosPolicy, 801
ASYNCHRONOUS_PUBLISHER, 561
DDS_ASYNCHRONOUSPUBLISHER_QOS_POLICY_NAME, 561
asynchronous_publisher
DDS_DiscoveryConfigQosPolicy, 949
DDS_PublisherQos, 1279
AsyncWaitSet, 488
COMPLETION_TOKEN_IGNORE, 489
COMPLETION_TOKEN_USE_IMPLICIT_AND_WAIT, 489
DDS_ASYNC_WAITSET_PROPERTY_DEFAULT, 489
attach_condition
DDSAsyncWaitSet, 1536
DDSWaitSet, 1902
attach_condition_with_completion_token
DDSAsyncWaitSet, 1537
autodispose_unregistered_instances
DDS_WriterDataLifecycleQosPolicy, 1525
autoenable_created_entities
DDS_EntityFactoryQosPolicy, 1147
autopurge_disposed_instances_delay
DDS_ReaderDataLifecycleQosPolicy, 1291
DDS_WriterDataLifecycleQosPolicy, 1526
autopurge_disposed_samples_delay
DDS_ReaderDataLifecycleQosPolicy, 1291
autopurge_nowriter_instances_delay
DDS_ReaderDataLifecycleQosPolicy, 1292
autopurge_nowriter_samples_delay
DDS_ReaderDataLifecycleQosPolicy, 1291
autopurge_remote_not_alive_writer_delay
DDS_DataReaderResourceLimitsQosPolicy, 884
autopurge_unregistered_instances_delay
DDS_WriterDataLifecycleQosPolicy, 1525
autoregister_instances
DDS_DataWriterResourceLimitsQosPolicy, 922
AVAILABILITY, 561
DDS_AVAILABILITY_QOS_POLICY_NAME, 562
availability
DDS_DataReaderQos, 865
DDS_DataWriterQos, 916
available_builtin_endpoints_ext
DDS_ParticipantBuiltinTopicData, 1231

banish_ignored_participants
DDSDomainParticipant, 1662
Basic QoS, 201
BATCH, 562
DDS_BATCH_QOS_POLICY_NAME, 563
batch
DDS_DataWriterQos, 916
Batching, 257
begin
connext::LoanedSamples< T >, 2003
rti::flat::AbstractAlignedList< ElementOffset >, 786
begin_access
DDSSubscriber, 1873
begin_coherent_changes
DDSPublisher, 1832
bind_complex_member
DDS_DynamicData, 1042
bind_type
DDS_DynamicData, 1041
binding_ping_period
NDDS_Transport_UDPv4_WAN_Property_t, 2076
bitmask
DDS_EndpointTrustProtectionInfo, 1145
DDS_ParticipantTrustProtectionInfo, 1238
bits
DDS_StructMember, 1366
DDS_ValueMember, 1507
buffer_alignment
DDS_ReceiverPoolQosPolicy, 1295
buffer_initial_size
DDS_DynamicDataProperty_t, 1137
buffer_max_size
DDS_DynamicDataProperty_t, 1137
buffer_size
DDS_ReceiverPoolQosPolicy, 1294
build
NDDS_Config_LibraryVersion_t, 2040
build_data
FlatData Builders, 768
build_my_final_seq
MyFlatMutableBuilder, 2023
build_my_mutable
MyFlatMutableBuilder, 2024
MyFlatUnionBuilder, 2034
build_my_mutable_array
MyFlatMutableBuilder, 2024
build_my_mutable_seq
MyFlatMutableBuilder, 2024
build_my_primitive_seq
MyFlatMutableBuilder, 2023
build_my_string
MyFlatMutableBuilder, 2024
build_my_string_seq
MyFlatMutableBuilder, 2024
build_next
rti::flat::MutableArrayBuilder< ElementBuilder, N >, 2009
rti::flat::MutableSequenceBuilder< ElementBuilder >, 2013
build_version
connext::MessagingLibraryVersion, 2006
Builder.hpp, 2287
BuilderHelper.hpp, 2295
Built-in Sequences, 169
Built-in Topic's Trust Types, 99
DDS_EndpointTrustAlgorithmInfo, 102
DDS_EndpointTrustAttributesMask, 100
DDS_EndpointTrustInterceptorAlgorithmInfo, 102
DDS_EndpointTrustProtectionInfo, 101
DDS_ParticipantTrustAlgorithmInfo, 102
DDS_ParticipantTrustAttributesMask, 100
DDS_ParticipantTrustInterceptorAlgorithmInfo, 102
DDS_ParticipantTrustKeyEstablishmentAlgorithmInfo, 102
DDS_ParticipantTrustProtectionInfo, 100
DDS_ParticipantTrustSignatureAlgorithmInfo, 101
DDS_PluginEndpointTrustAttributesMask, 101
DDS_PluginParticipantTrustAttributesMask, 100
DDS_TrustAlgorithmBit, 101
DDS_TrustAlgorithmRequirements, 101
DDS_TrustAlgorithmSet, 101
DDS_VendorEndpointTrustAttributesMask, 101
Built-in Topics, 61
Built-in Transport Plugins, 182
Built-in Types, 95
Builtin Qos Profiles, 679
DDS_BUILTIN_QOS_LIB, 685
DDS_BUILTIN_QOS_LIB_EXP, 689
DDS_BUILTIN_QOS_SNIPPET_LIB, 700
DDS_PROFILE_BASELINE, 685
DDS_PROFILE_BASELINE_5_0_0, 685
DDS_PROFILE_BASELINE_5_1_0, 685
DDS_PROFILE_BASELINE_5_2_0, 686
DDS_PROFILE_BASELINE_5_3_0, 686
DDS_PROFILE_BASELINE_6_0_0, 686
DDS_PROFILE_BASELINE_6_1_0, 686
DDS_PROFILE_BASELINE_7_0_0, 686
DDS_PROFILE_BASELINE_7_1_0, 687
DDS_PROFILE_BASELINE_ROOT, 685
DDS_PROFILE_GENERIC_510_TRANSPORT_COMPATIBILITY, 688
DDS_PROFILE_GENERIC_AUTO_TUNING, 694
DDS_PROFILE_GENERIC_BEST_EFFORT, 689
DDS_PROFILE_GENERIC_COMMON, 687
DDS_PROFILE_GENERIC_CONNEXT_MICRO_COMPATIBILITY, 687
DDS_PROFILE_GENERIC_CONNEXT_MICRO_COMPATIBILITY_2_4 688

DDS_PROFILE_GENERIC_CONNEXT_MICRO_COMPATIBILITY_PROFILE, 688
DDS_PROFILE_GENERIC_KEEP_LAST_RELIABLE, 689
DDS_PROFILE_GENERIC_KEEP_LAST_RELIABLE_LARGE_DATA, 692
DDS_PROFILE_GENERIC_KEEP_LAST_RELIABLE_LARGE_DATA_FAST_FLOW, 693
DDS_PROFILE_GENERIC_KEEP_LAST_RELIABLE_LARGE_DATA_MEDIUM_FLOW, 693
DDS_PROFILE_GENERIC_KEEP_LAST_RELIABLE_LARGE_DATA_SLOW_FLOW, 693
DDS_PROFILE_GENERIC_KEEP_LAST_RELIABLE_PERSISTENT, 694
DDS_PROFILE_GENERIC_KEEP_LAST_RELIABLE_TRANSIENT, 694
DDS_PROFILE_GENERIC_KEEP_LAST_RELIABLE_TRANSIENT_LOCAL, 694
DDS_PROFILE_GENERIC_MINIMAL_MEMORY_FOOTPRINT, 695
DDS_PROFILE_GENERIC_MONITORING2, 695
DDS_PROFILE_GENERIC_MONITORING2_WAN, 696
DDS_PROFILE_GENERIC_MONITORING_COMMON, 687
DDS_PROFILE_GENERIC_OTHER_DDS_VENDOR_COMPATIBILITY, 688
DDS_PROFILE_GENERIC_PARTICIPANT_LARGE_DATA, 690
DDS_PROFILE_GENERIC_PARTICIPANT_LARGE_DATA_MONITORING, 691
DDS_PROFILE_GENERIC_SECURITY, 695
DDS_PROFILE_GENERIC_STRICT_RELIABLE, 689
DDS_PROFILE_GENERIC_STRICT_RELIABLE_HIGH_THROUGHPUT, 690
DDS_PROFILE_GENERIC_STRICT_RELIABLE_LARGE_DATA, 691
DDS_PROFILE_GENERIC_STRICT_RELIABLE_LARGE_DATA_FAST_FLOW, 692
DDS_PROFILE_GENERIC_STRICT_RELIABLE_LARGE_DATA_MEDIUM_FLOW, 692
DDS_PROFILE_GENERIC_STRICT_RELIABLE_LARGE_DATA_SLOW_FLOW, 693
DDS_PROFILE_GENERIC_STRICT_RELIABLE_LOW_LATENCY, 690
DDS_PROFILE_PATTERN_ALARM_EVENT, 698
DDS_PROFILE_PATTERN_ALARM_STATUS, 699
DDS_PROFILE_PATTERN_EVENT, 698
DDS_PROFILE_PATTERN_LAST_VALUE_CACHE, 699
DDS_PROFILE_PATTERN_PERIODIC_DATA, 697
DDS_PROFILE_PATTERN_RELIABLE_STREAMING, 698
DDS_PROFILE_PATTERN_RPC, 697
DDS_PROFILE_PATTERN_STATUS, 699
DDS_PROFILE_PATTERN_STREAMING, 697
DDS_SNIPPET_5_1_0_TRANSPORT_ENABLE, 716
DDS_SNIPPET_COMPATIBILITY_CONNEXT_MICRO_VERSION_2_4, 715
DDS_SNIPPET_COMPATIBILITY_OTHER_DDS_VENDORS_ENABLE, 710
DDS_SNIPPET_FEATURE_AUTO_TUNING_ENABLE, 710
DDS_SNIPPET_FEATURE_FLOW_CONTROLLER_209MBPS, 709
DDS_SNIPPET_FEATURE_FLOW_CONTROLLER_52MBPS, 709
DDS_SNIPPET_FEATURE_FLOW_CONTROLLER_838MBPS, 709
DDS_SNIPPET_FEATURE_MONITORING2_ENABLE, 711
DDS_SNIPPET_FEATURE_MONITORING_ENABLE, 710
DDS_SNIPPET_FEATURE_SECURITY_ENABLE, 711
DDS_SNIPPET_FEATURE_TOPIC_QUERY_ENABLE, 711
DDS_SNIPPET_FEATURE_TYPE_SUPPORT_V1_DISABLE, 712
DDS_SNIPPET_FEATURE_TYPE_SUPPORT_V1_ENABLE, 712
DDS_SNIPPET_FEATURE_TYPE_SUPPORT_V2_DISABLE, 713
DDS_SNIPPET_FEATURE_TYPE_SUPPORT_V2_ENABLE, 712
DDS_SNIPPET_OPTIMIZATION_DATACACHE_LARGE_DATA_DYNAMIC, 703
DDS_SNIPPET_OPTIMIZATION_DISCOVERY_COMMON, 703
DDS_SNIPPET_OPTIMIZATION_DISCOVERY_ENDPOINT_FAST, 704
DDS_SNIPPET_OPTIMIZATION_DISCOVERY_PARTICIPANT_COMPACT, 704
DDS_SNIPPET_OPTIMIZATION_RELIABILITY_PROTOCOL_COMMON, 700
DDS_SNIPPET_OPTIMIZATION_RELIABILITY_PROTOCOL_HIGH_RATE, 701
DDS_SNIPPET_OPTIMIZATION_RELIABILITY_PROTOCOL_KEEP_ALL, 700
DDS_SNIPPET_OPTIMIZATION_RELIABILITY_PROTOCOL_KEEP_LAST, 701
DDS_SNIPPET_OPTIMIZATION_RELIABILITY_PROTOCOL_LARGE_DATA, 702
DDS_SNIPPET_OPTIMIZATION_RELIABILITY_PROTOCOL_LOW_LATENCY, 702
DDS_SNIPPET_OPTIMIZATION_TRANSPORT_LARGE_BUFFERS,

705
DDS_SNIPPET_QOS_POLICY_BATCHING_ENABLE, 708
DDS_SNIPPET_QOS_POLICY_DURABILITY_PERSISTENT, 708
DDS_SNIPPET_QOS_POLICY_DURABILITY_TRANSIENT, 707
DDS_SNIPPET_QOS_POLICY_DURABILITY_TRANSIENT_LOCAL, 707
DDS_SNIPPET_QOS_POLICY_HISTORY_KEEP_ALL, 706
DDS_SNIPPET_QOS_POLICY_HISTORY_KEEP_LAST, 706
DDS_SNIPPET_QOS_POLICY_PUBLISH_MODE_ASYNCHRONOUS, 706
DDS_SNIPPET_QOS_POLICY_RELIABILITY_BEST_EFFORT, 705
DDS_SNIPPET_QOS_POLICY_RELIABILITY_RELIABLE, 705
DDS_SNIPPET_TRANSPORT_TCP_LAN_CLIENT, 713
DDS_SNIPPET_TRANSPORT_TCP_WAN_ASYMMETRIC_CLIENT, 714
DDS_SNIPPET_TRANSPORT_TCP_WAN_ASYMMETRIC_SERVER, 714
DDS_SNIPPET_TRANSPORT_TCP_WAN_SYMMETRIC_CLIENT, 713
DDS_SNIPPET_TRANSPORT_UDP_AVOID_IP_FRAGMENTATION, 714
DDS_SNIPPET_TRANSPORT_UDP_WAN, 715
builtin_discovery_plugins
DDS_DiscoveryConfigQosPolicy, 946
builtin_endpoints_required_mask
DDS_ParticipantTrustInterceptorAlgorithmInfo, 1237
builtin_kx_endpoints_required_mask
DDS_ParticipantTrustInterceptorAlgorithmInfo, 1237
builtin_multicast_port_offset
DDS_RtpsWellKnownPorts_t, 1342
builtin_unicast_port_offset
DDS_RtpsWellKnownPorts_t, 1342
bytes_per_token
DDS_FlowControllerTokenBucketProperty_t, 1159

capacity
rti::flat::AbstractBuilder, 788
category
DDS_LoggingQosPolicy, 1189
cdr_padding_kind
DDS_TypeSupportQosPolicy, 1500
cdr_serialized_sample_key_max_size
DDS_TypeCode, 1459, 1460
cdr_serialized_sample_max_size
DDS_TypeCode, 1458, 1459
cdr_serialized_sample_min_size
DDS_TypeCode, 1458, 1460
channel_filter_expression_max_length
DDS_DomainParticipantResourceLimitsQosPolicy, 998
channel_seq_max_length
DDS_DomainParticipantResourceLimitsQosPolicy, 997
channels
DDS_MultiChannelQosPolicy, 1215
check_crc
DDS_WireProtocolQosPolicy, 1518
check_failure
rti::flat::AbstractBuilder, 789
checkpoint_thread_settings
NDDS_Utility_NetworkCaptureParams_t, 2088
class_id
DDS_TransportInfo_t, 1409
classid
NDDS_Transport_Property_t, 2045
cleanup_period
DDS_DatabaseQosPolicy, 833
clear_all_members
DDS_DynamicData, 1033
clear_member
DDS_DynamicData, 1034
clear_optional_member
DDS_DynamicData, 1033
clock_selection
DDS_TypeCode, 1458, 1460
clone
rti::flat::Sample< OffsetType >, 2190
clone_tc
DDS_TypeCodeFactory, 1479
close
NDDSConfigLoggerDevice, 2101
coherent_access
DDS_PresentationQosPolicy, 1252
coherent_set_info
DDS_SampleInfo, 1354
coherent_set_sequence_number
DDS_CoherentSetInfo_t, 825
collector_initial_peers
DDS_MonitoringDedicatedParticipantSettings, 1194
comm_ports_list
NDDS_Transport_UDPv4_WAN_Property_t, 2074
comm_ports_list_length
NDDS_Transport_UDPv4_WAN_Property_t, 2075
Common types and functions, 499
DDS_BUILTIN_TOPIC_KEY_UNKNOWN, 508
DDS_BuiltinTopicKey_copy, 504
DDS_BuiltinTopicKey_equals, 504
DDS_BuiltinTopicKey_from_guid, 505
DDS_BuiltinTopicKey_from_instance_handle, 506
DDS_BuiltinTopicKey_t, 503
DDS_BuiltinTopicKey_to_guid, 505

DDS_BuiltinTopicKey_to_instance_handle, 505
DDS_LOCATOR_ADDRESS_INVALID, 507
DDS_LOCATOR_ADDRESS_LENGTH_MAX, 501
DDS_LOCATOR_INVALID, 506
DDS_LOCATOR_KIND_INVALID, 506
DDS_LOCATOR_KIND_RESERVED, 508
DDS_LOCATOR_KIND_SHMEM, 507
DDS_LOCATOR_KIND_SHMEM_510, 507
DDS_LOCATOR_KIND_UDPv4, 507
DDS_LOCATOR_KIND_UDPv4_WAN, 507
DDS_LOCATOR_KIND_UDPv6, 507
DDS_LOCATOR_KIND_UDPv6_510, 508
DDS_LOCATOR_PORT_INVALID, 506
DDS_Locator_t, 503
DDS_PRODUCTVERSION_UNKNOWN, 503
DDS_PROTOCOLVERSION, 502
DDS_PROTOCOLVERSION_1_0, 501
DDS_PROTOCOLVERSION_1_1, 502
DDS_PROTOCOLVERSION_1_2, 502
DDS_PROTOCOLVERSION_2_0, 502
DDS_PROTOCOLVERSION_2_1, 502
DDS_ProtocolVersion_t, 503
DDS_VENDOR_ID_LENGTH_MAX, 502
value, 508
compile
DDSContentFilter, 1548
COMPLETION_TOKEN_IGNORE
AsyncWaitSet, 489
COMPLETION_TOKEN_USE_IMPLICIT_AND_WAIT
AsyncWaitSet, 489
Compliance Configuration, 734
NDDS_CONFIG_RPC_COMPLIANCE_MASK_DEFAULT, 736
NDDS_CONFIG_RPC_COMPLIANCE_MASK_VENDOR, 736
NDDS_CONFIG_RPC_USE_STANDARD_RELATED_SAMPLE_ID_BIT, 745
NDDS_Config_RpcComplianceMask, 737
NDDS_Config_RpcComplianceMaskBits, 745
NDDS_CONFIG_XTYPES_ACCEPT_UNKNOWN_DISCRIMINATOR_BIT, 741
NDDS_CONFIG_XTYPES_ACCEPT_UNKNOWN_ENUM_VALUE_BIT, 740
NDDS_CONFIG_XTYPES_COMPLIANCE_MASK_DEFAULT, 735
NDDS_CONFIG_XTYPES_COMPLIANCE_MASK_VENDOR, 735
NDDS_CONFIG_XTYPES_DHEADER_IN_NON_PRIMITIVE_TYPES_BIT, 739
NDDS_CONFIG_XTYPES_ENCAPSULATION_OPTIONS_BIT, 740
NDDS_CONFIG_XTYPES_ENUM_AS_PRIMITIVE_IN_CONNEXT_BIT, 739
NDDS_CONFIG_XTYPES_INITIALIZE_DISCRIMINATOR_TO_DEFAULT_BIT, 743
NDDS_CONFIG_XTYPES_JAVA_KEYHASH_750_BIT, 744
NDDS_CONFIG_XTYPES_MUST_UNDERSTAND_IN_KEY_AND_DIS 745
NDDS_CONFIG_XTYPES_PARAMETER_LENGTH_WITH_PADDING_ 739
NDDS_CONFIG_XTYPES_PYTHON_INT8_730_BIT, 744
NDDS_CONFIG_XTYPES_SELECT_DEFAULT_DISCRIMINATOR_BIT 741
NDDS_CONFIG_XTYPES_SENTINEL_IN_EMPTY_UNION_BIT, 742
NDDS_Config_XTypesComplianceMask, 736
NDDS_Config_XTypesComplianceMaskBits, 738
compressed_sample_count
DDS_DataReaderCacheStatus, 842
Compression Settings, 570
DDS_COMPRESSION_ID_BZIP2, 573
DDS_COMPRESSION_ID_LZ4, 574
DDS_COMPRESSION_ID_MASK_ALL, 571
DDS_COMPRESSION_ID_MASK_NONE, 571
DDS_COMPRESSION_ID_MASK_PUBLICATION_DEFAULT, 572
DDS_COMPRESSION_ID_MASK_SUBSCRIPTION_DEFAULT, 572
DDS_COMPRESSION_ID_ZLIB, 573
DDS_COMPRESSION_LEVEL_BEST_COMPRESSION, 571
DDS_COMPRESSION_LEVEL_BEST_SPEED, 571
DDS_COMPRESSION_LEVEL_DEFAULT, 572
DDS_COMPRESSION_THRESHOLD_DEFAULT, 572
DDS_CompressionId_t, 573
DDS_CompressionIdMask, 573
compression_id, 573
DDS_CompressionSettings_t, 826
compression_settings
DDS_DataRepresentationQosPolicy, 888
COMPRESSION_BIT, 1518
DDS_WireProtocolQosPolicy, 1518
compression_type
DDS_TypeCode, 1453
Concurrency, 258
concurrency_level
DDS_MonitoringEventDistributionSettings, 1198
DDS_MonitoringLoggingDistributionSettings, 1201
CONDITION_LIST_MAX_SIZE, 670
DDS_WaitSetProperty_t_INITIALIZER, 670
Configuring QoS Profiles with XML, 390
connext, 779
connext::AlreadyDeletedException, 793
connext::BadParameterException, 794
connext::IllegalOperationException, 1992
CONNECTIONS_BIT, 1992

connext::ImmutablePolicyException, 1992
connext::InconsistentPolicyException, 1993
connext::IsInvalidSamplePredicate< T >, 1993
connext::IsReplyRelatedPredicate< T >, 1993
    IsReplyRelatedPredicate, 1994
    operator(), 1994
connext::IsValidSamplePredicate< T >, 1995
connext::LoanedSamples< T >, 1995
    ~LoanedSamples, 1999
    begin, 2003
    const_iterator, 1998
    end, 2003, 2004
    iterator, 1997
    length, 2002
    LoanedSamples, 1998
    move_construct_from_loans, 1999
    operator LoanMemento, 2002
    operator[], 2001
    release, 2000
    return_loan, 2002
connext::LoanedSamples< T >::LoanMemento, 2004
connext::LogicException, 2005
connext::MessagingLibraryVersion, 2005
    build_version, 2006
    major_version, 2006
    minor_version, 2006
    release_version, 2006
connext::MessagingVersion, 2006
    get_api_build_version, 2007
    get_api_version, 2007
    get_api_version_string, 2007
connext::NotEnabledException, 2126
connext::OutOfResourcesException, 2131
connext::PreconditionNotMetException, 2132
connext::Replier< TReq, TRep >, 2139
    ~Replier, 2144
    get_reply_datawriter, 2151
    get_request_datareader, 2150
    read_request, 2150
    read_requests, 2150
    receive_request, 2146, 2147
    receive_requests, 2147
    Replier, 2143, 2144
    Reply, 2142
    ReplyDataWriter, 2142
    Request, 2142
    RequestDataReader, 2142
    send_reply, 2145, 2146
    take_request, 2149
    take_requests, 2149
    wait_for_requests, 2148
connext::ReplierListener< TReq, TRep >, 2151
    on_request_available, 2152
connext::ReplierParams< TReq, TRep >, 2152
    datareader_qos, 2155
    datawriter_qos, 2155
    publisher, 2155
    qos_profile, 2154
    replier_listener, 2153
    ReplierParams, 2153
    reply_topic_name, 2154
    reply_type_support, 2156
    request_topic_name, 2154
    request_type_support, 2156
    service_name, 2154
    subscriber, 2156
connext::Requester< TReq, TRep >, 2157
    ~Requester, 2162
    get_reply_datareader, 2177
    get_request_datawriter, 2177
    read_replies, 2175, 2176
    read_reply, 2174–2176
    receive_replies, 2165, 2166
    receive_reply, 2164, 2165
    Reply, 2160
    ReplyDataReader, 2161
    ReplyTypeSupport, 2161
    Request, 2160
    RequestDataWriter, 2161
    Requester, 2161, 2162
    RequestTypeSupport, 2161
    send_request, 2163, 2164
    take_replies, 2170, 2172, 2174
    take_reply, 2169, 2171, 2172
    wait_for_replies, 2167, 2168
connext::RequesterParams, 2178
    datareader_qos, 2181
    datawriter_qos, 2181
    publisher, 2181
    qos_profile, 2180
    reply_topic_name, 2180
    reply_type_support, 2182
    request_topic_name, 2180
    request_type_support, 2182
    RequesterParams, 2179
    service_name, 2180
    subscriber, 2182
connext::RuntimeException, 2183
connext::Sample< T >, 2183
    data, 2185, 2186
    identity, 2185
    info, 2186
    related_identity, 2185
    Sample, 2184
connext::SampleIterator< T, IsConst >, 2191
    operator++, 2191
connext::SampleRef< T >, 2191
    data, 2194

identity, 2197
info, 2195
is_nil_data, 2196
is_nil_info, 2197
operator T&, 2195
operator=, 2194
related_identity, 2197
SampleRef, 2193, 2194
set_data, 2196
set_info, 2196
connext::SimpleReplier< TReq, TRep >, 2206
~SimpleReplier, 2207
SimpleReplier, 2207
connext::SimpleReplierListener< TReq, TRep >, 2208
on_request_available, 2208
return_loan, 2209
connext::SimpleReplierParams< TReq, TRep >, 2209
datareader_qos, 2212
datawriter_qos, 2212
publisher, 2212
qos_profile, 2211
reply_topic_name, 2211
reply_type_support, 2213
request_topic_name, 2211
request_type_support, 2213
service_name, 2211
SimpleReplierParams, 2210
subscriber, 2213
connext::TimeoutException, 2217
connext::UnsupportedException, 2219
connext::WriteSample< T >, 2219
data, 2221, 2222
identity, 2222
info, 2222
WriteSample, 2220, 2221
connext::WriteSampleRef< T >, 2223
data, 2225
identity, 2226
info, 2225
is_nil_data, 2226
set_data, 2225
set_info, 2226
WriteSampleRef, 2224
const_iterator
connext::LoanedSamples< T >, 1998
ConstOffset
MyFlatFinalOffset, 2015
MyFlatMutableOffset, 2026
MyFlatUnionOffset, 2036
rti::flat::Sample< OffsetType >, 2188
contains_entity
DDSDomainParticipant, 1674
Content Filtering, 260
content_filter_allocation
DDS_DomainParticipantResourceLimitsQosPolicy, 984
content_filter_dropped_sample_count
DDS_DataReaderCacheStatus, 838
content_filter_hash_buckets
DDS_DomainParticipantResourceLimitsQosPolicy, 990
content_filter_property
DDS_SubscriptionBuiltinTopicData, 1377
content_filter_topic_name
DDS_ContentFilterProperty_t, 829
content_filtered_topic_allocation
DDS_DomainParticipantResourceLimitsQosPolicy, 984
content_filtered_topic_hash_buckets
DDS_DomainParticipantResourceLimitsQosPolicy, 990
content_type
DDS_TypeCode, 1451
contentfilter_property_max_length
DDS_DomainParticipantResourceLimitsQosPolicy, 997
Conventions, 431
Cookie, 671
to_pointer, 671
cookie
DDS_WriteParams_t, 1521
copy
DDS_DynamicData, 1031
copy_data
DDSDynamicDataTypeSupport, 1729
FooTypeSupport, 1984
copy_from_topic_qos
DDSPublisher, 1834
DDSSubscriber, 1876
copy_no_alloc
FooSeq, 1970
corrupted_rtps_message_count
DDS_DomainParticipantProtocolStatus, 970
corrupted_rtps_message_count_change
DDS_DomainParticipantProtocolStatus, 970
count
DDS_QosPolicyCount, 1284
cpu_list
DDS_ThreadSettings_t, 1386
cpu_rotation
DDS_ThreadSettings_t, 1387
create_alias_tc
DDS_TypeCodeFactory, 1486
create_array_tc
DDS_TypeCodeFactory, 1489
create_completion_token
DDSAsyncWaitSet, 1540
create_contentfilteredtopic

DDSDomainParticipant, 1653
create_contentfilteredtopic_with_filter
DDSDomainParticipant, 1654
create_data
DDSDynamicDataTypeSupport, 1728
DDSKeyedOctetsDataWriter, 1756
DDSKeyedStringDataWriter, 1783
DDSOctetsDataWriter, 1806
DDSStringDataWriter, 1853
FooDataWriter, 1962
FooTypeSupport, 1983
rti::flat::Sample< OffsetType >, 2189
create_data_ex
FooTypeSupport, 1983
create_datareader
DDSDomainParticipant, 1687
DDSSubscriber, 1867
create_datareader_with_profile
DDSDomainParticipant, 1688
DDSSubscriber, 1869
create_datawriter
DDSDomainParticipant, 1683
DDSPublisher, 1826
create_datawriter_with_profile
DDSDomainParticipant, 1685
DDSPublisher, 1827
create_enum_tc
DDS_TypeCodeFactory, 1485, 1486
create_flowcontroller
DDSDomainParticipant, 1659
create_multitopic
DDSDomainParticipant, 1656
create_participant
DDSDomainParticipantFactory, 1710
create_participant_from_config
DDSDomainParticipantFactory, 1717
create_participant_from_config_w_params
DDSDomainParticipantFactory, 1718
create_participant_with_profile
DDSDomainParticipantFactory, 1711
create_publisher
DDSDomainParticipant, 1644
create_publisher_with_profile
DDSDomainParticipant, 1645
create_querycondition
DDSDataReader, 1561
create_querycondition_w_params
DDSDataReader, 1562
create_readcondition
DDSDataReader, 1561
create_readcondition_w_params
DDSDataReader, 1561
create_sequence_tc
DDS_TypeCodeFactory, 1488
create_string_tc
DDS_TypeCodeFactory, 1487
create_struct_tc
DDS_TypeCodeFactory, 1480, 1481
create_subscriber
DDSDomainParticipant, 1647
create_subscriber_with_profile
DDSDomainParticipant, 1648
create_thread
DDSThreadFactory, 1884
create_topic
DDSDomainParticipant, 1650
create_topic_query
DDSDataReader, 1577
create_topic_with_profile
DDSDomainParticipant, 1651
create_union_tc
DDS_TypeCodeFactory, 1483, 1484
create_value_tc
DDS_TypeCodeFactory, 1482, 1483
create_wstring_tc
DDS_TypeCodeFactory, 1487
Creating Custom Content Filters, 415
Creating New Transport Plugins, 183
critical
NDDSConfigLogger, 2099
current_count
DDS_PublicationMatchedStatus, 1275
DDS_ServiceRequestAcceptedStatus, 1362
DDS_SubscriptionMatchedStatus, 1381
current_count_change
DDS_PublicationMatchedStatus, 1275
DDS_ServiceRequestAcceptedStatus, 1362
DDS_SubscriptionMatchedStatus, 1381
current_count_peak
DDS_PublicationMatchedStatus, 1275
DDS_SubscriptionMatchedStatus, 1381

data
connext::Sample< T >, 2185, 2186
connext::SampleRef< T >, 2194
connext::WriteSample< T >, 2221, 2222
connext::WriteSampleRef< T >, 2225
DDS_DynamicDataTypeProperty_t, 1139
Data Availability, 261
Data Samples, 152
DDS_CoherentSetInfo_copy, 154
DDS_CoherentSetInfo_equals, 153
DDS_SampleInfo_get_related_sample_identity, 154
DDS_SampleInfo_get_sample_identity, 154
Data Writers, 115
DDS_DataWriterQos_equals, 117
print, 117
to_string, 117–121

DATA_READER_PROTOCOL, 564
DDS_DATAREADERPROTOCOL_QOS_POLICY_NAME, 564
DATA_READER_RESOURCE_LIMITS, 564
DDS_ANY_INSTANCE_REMOVAL, 566
DDS_AUTO_MAX_TOTAL_INSTANCES, 567
DDS_DataReaderInstanceRemovalKind, 565
DDS_DATAREADERRESOURCELIMITS_QOS_POLICY_NAME, 567
DDS_EMPTY_INSTANCE_REMOVAL, 566
DDS_FULLY_PROCESSED_INSTANCE_REMOVAL, 566
DDS_NO_INSTANCE_REMOVAL, 566
DATA_REPRESENTATION, 567
DDS_AUTO_DATA_REPRESENTATION, 569
DDS_DATA_REPRESENTATION_QOS_POLICY_NAME, 569
DDS_DataRepresentationId_t, 568
DDS_XCDR2_DATA_REPRESENTATION, 569
DDS_XCDR_DATA_REPRESENTATION, 568
DDS_XML_DATA_REPRESENTATION, 568
DATA_TAG, 574
add_tag, 577
assert_tag, 576
DDS_DATATAG_QOS_POLICY_NAME, 579
DDS_DataTagQosPolicy, 575
get_number_of_tags, 576
lookup_tag, 577
remove_tag, 578
data_tags
DDS_DataReaderQos, 863
DDS_DataWriterQos, 915
DDS_PublicationBuiltinTopicData, 1269
DDS_SubscriptionBuiltinTopicData, 1375
data_to_string
DDSKeyedOctetsTypeSupport, 1769
DDSKeyedStringTypeSupport, 1794
DDSOctetsTypeSupport, 1815
DDSStringTypeSupport, 1859
FooTypeSupport, 1991
DATA_WRITER_PROTOCOL, 579
DDS_DATAWRITERPROTOCOL_QOS_POLICY_NAME, 579
DATA_WRITER_RESOURCE_LIMITS, 580
DDS_ALIVE_INSTANCE_REPLACEMENT, 582
DDS_ALIVE_OR_DISPOSED_INSTANCE_REPLACEMENT, 582
DDS_ALIVE_THEN_DISPOSED_INSTANCE_REPLACEMENT, 582
DDS_DATAWRITERRESOURCELIMITS_QOS_POLICY_NAME, 582
DDS_DataWriterResourceLimitsInstanceReplacementKind, 580
DDS_DISPOSED_INSTANCE_REPLACEMENT, 582
DDS_DISPOSED_THEN_ALIVE_INSTANCE_REPLACEMENT, 582
DDS_UNREGISTERED_INSTANCE_REPLACEMENT, 581
DATA_WRITER_TRANSFER_MODE, 583
DDS_DATAWRITERTRANSFERMODE_QOS_POLICY_NAME, 583
DATABASE, 563
DDS_DATABASE_QOS_POLICY_NAME, 563
database
DDS_DomainParticipantQos, 975
DataReader Use Cases, 403
datareader_qos
connext::ReplierParams< TReq, TRep >, 2155
connext::RequesterParams, 2181
connext::SimpleReplierParams< TReq, TRep >, 2212
DataReaders, 138
DDS_DataReaderQos_equals, 146
DDS_LOST_BY_AVAILABILITY_WAITING_TIME, 142
DDS_LOST_BY_DECODE_FAILURE, 143
DDS_LOST_BY_DESERIALIZATION_FAILURE, 143
DDS_LOST_BY_INCOMPLETE_COHERENT_SET, 141
DDS_LOST_BY_INSTANCES_LIMIT, 140
DDS_LOST_BY_LARGE_COHERENT_SET, 141
DDS_LOST_BY_OUT_OF_MEMORY, 143
DDS_LOST_BY_REMOTE_WRITER_SAMPLES_PER_VIRTUAL_QUE 143
DDS_LOST_BY_REMOTE_WRITERS_PER_INSTANCE_LIMIT, 140
DDS_LOST_BY_REMOTE_WRITERS_PER_SAMPLE_LIMIT, 142
DDS_LOST_BY_SAMPLES_LIMIT, 144
DDS_LOST_BY_SAMPLES_PER_INSTANCE_LIMIT, 143
DDS_LOST_BY_SAMPLES_PER_REMOTE_WRITER_LIMIT, 142
DDS_LOST_BY_UNKNOWN_INSTANCE, 143
DDS_LOST_BY_VIRTUAL_WRITERS_LIMIT, 142
DDS_LOST_BY_WRITER, 140
DDS_NOT_LOST, 140
DDS_NOT_REJECTED, 144
DDS_REJECTED_BY_DECODE_FAILURE, 146
DDS_REJECTED_BY_INSTANCES_LIMIT, 144
DDS_REJECTED_BY_REMOTE_WRITER_SAMPLES_PER_VIRTUAL
DDS_REJECTED_BY_SAMPLES_LIMIT, 144
DDS_REJECTED_BY_SAMPLES_PER_INSTANCE_LIMIT, 145
DDS_REJECTED_BY_SAMPLES_PER_REMOTE_WRITER_LIMIT,

145
DDS_SampleLostStatusKind, 140
DDS_SampleRejectedStatusKind, 144
print, 146
to_string, 147–150
DataWriter Use Cases, 400
datawriter_qos
connext::ReplierParams< TReq, TRep >, 2155
connext::RequesterParams, 2181
connext::SimpleReplierParams< TReq, TRep >, 2212
datawriter_qos_profile_name
DDS_MonitoringEventDistributionSettings, 1199
DDS_MonitoringLoggingDistributionSettings, 1202
DDS_MonitoringPeriodicDistributionSettings, 1209
DDS Extension QoS, 213
DDS Specification QoS, 209
DDS-Specific Primitive Types, 511
DDS_Boolean, 516
DDS_BOOLEAN_FALSE, 513
DDS_BOOLEAN_TRUE, 513
DDS_Char, 513
DDS_Double, 515
DDS_Enum, 516
DDS_Float, 515
DDS_Int8, 514
DDS_Long, 515
DDS_LongDouble, 516
DDS_LongLong, 515
DDS_Octet, 514
DDS_Short, 514
DDS_UInt8, 514
DDS_UnsignedLong, 515
DDS_UnsignedLongLong, 515
DDS_UnsignedShort, 514
DDS_Wchar, 514
DDS_AcknowledgmentInfo, 794
response_data, 795
sample_identity, 795
subscription_handle, 795
valid_response_data, 795
DDS_AckResponseData_t, 796
value, 796
DDS_ADMIN_CONSOLE_SERVICE_QOS
SERVICE, 637
DDS_ALIVE_INSTANCE_REPLACEMENT
DATA_WRITER_RESOURCE_LIMITS, 582
DDS_ALIVE_INSTANCE_STATE
Instance States, 166
DDS_ALIVE_OR_DISPOSED_INSTANCE_REPLACEMENT
DATA_WRITER_RESOURCE_LIMITS, 582
DDS_ALIVE_THEN_DISPOSED_INSTANCE_REPLACEMENT
DATA_WRITER_RESOURCE_LIMITS, 582
DDS_AllocationSettings_t, 796
incremental_count, 797
initial_count, 797
max_count, 797
DDS_ALLOW_TYPE_COERCION
TYPE_CONSISTENCY_ENFORCEMENT, 651
DDS_AnnotationParameterValue, 798
DDS_ANY_INSTANCE_REMOVAL
DATA_READER_RESOURCE_LIMITS, 566
DDS_ANY_INSTANCE_STATE
Instance States, 166
DDS_ANY_SAMPLE_STATE
Sample States, 162
DDS_ANY_VIEW_STATE
View States, 164
DDS_APPLICATION_AUTO_ACKNOWLEDGMENT_MODE
RELIABILITY, 633
DDS_APPLICATION_EXPLICIT_ACKNOWLEDGMENT_MODE
RELIABILITY, 634
DDS_ASYNC_WAITSET_PROPERTY_DEFAULT
AsyncWaitSet, 489
DDS_ASYNCHRONOUS_PUBLISH_MODE_QOS
PUBLISH_MODE, 629
DDS_ASYNCHRONOUSPUBLISHER_QOS_POLICY_ID
All QoS Policies, 559
DDS_ASYNCHRONOUSPUBLISHER_QOS_POLICY_NAME
ASYNCHRONOUS_PUBLISHER, 561
DDS_AsynchronousPublisherQosPolicy, 798
asynchronous_batch_thread, 801
disable_asynchronous_batch, 800
disable_asynchronous_write, 800
disable_topic_query_publication, 801
thread, 800
topic_query_publication_thread, 802
DDS_AsyncWaitSetProperty_t, 802
level, 805
thread_name_prefix, 804
thread_pool_size, 803
thread_settings, 804
wait_timeout, 804
waitset_property, 803
DDS_AUTO_CDR_PADDING
TYPESUPPORT, 652
DDS_AUTO_COUNT
DOMAIN_PARTICIPANT_RESOURCE_LIMITS, 595
DDS_AUTO_DATA_REPRESENTATION
DATA_REPRESENTATION, 569
DDS_AUTO_MAX_TOTAL_INSTANCES
DATA_READER_RESOURCE_LIMITS, 567
DDS_AUTO_SAMPLE_IDENTITY
WriteParams, 676
DDS_AUTO_SEQUENCE_NUMBER
Sequence Number Support, 529
DDS_AUTO_TYPE_COERCION
TYPE_CONSISTENCY_ENFORCEMENT, 651

DDS_AUTO_WRITER_DEPTH
DURABILITY, 599
DDS_AUTOMATIC_LIVELINESS_QOS
LIVELINESS, 608
DDS_AUTOMATIC_TRANSPORT_MULTICAST_QOS
TRANSPORT_MULTICAST, 646
DDS_AVAILABILITY_QOS_POLICY_ID
All QoS Policies, 559
DDS_AVAILABILITY_QOS_POLICY_NAME
AVAILABILITY, 562
DDS_AvailabilityQosPolicy, 805
enable_required_subscriptions, 808
max_data_availability_waiting_time, 808
max_endpoint_availability_waiting_time, 808
required_matched_endpoint_groups, 809
DDS_BAD_MEMBER_ID_USER_EXCEPTION_CODE
Exception Codes, 531
DDS_BAD_MEMBER_NAME_USER_EXCEPTION_CODE
Exception Codes, 531
DDS_BAD_PARAM_SYSTEM_EXCEPTION_CODE
Exception Codes, 530
DDS_BAD_TYPECODE_SYSTEM_EXCEPTION_CODE
Exception Codes, 530
DDS_BADKIND_USER_EXCEPTION_CODE
Exception Codes, 530
DDS_BATCH_QOS_POLICY_ID
All QoS Policies, 559
DDS_BATCH_QOS_POLICY_NAME
BATCH, 563
DDS_BatchQosPolicy, 810
enable, 810
max_data_bytes, 811
max_flush_delay, 812
max_samples, 811
source_timestamp_resolution, 812
thread_safe_write, 813
DDS_BEST_EFFORT_RELIABILITY_QOS
RELIABILITY, 633
DDS_Boolean
DDS-Specific Primitive Types, 516
DDS_BOOLEAN_FALSE
DDS-Specific Primitive Types, 513
DDS_BOOLEAN_TRUE
DDS-Specific Primitive Types, 513
DDS_BooleanSeq, 814
DDS_BOUNDS_USER_EXCEPTION_CODE
Exception Codes, 530
dds_builtin_endpoints
DDS_ParticipantBuiltinTopicData, 1231
DDS_BUILTIN_QOS_LIB
Builtin Qos Profiles, 685
DDS_BUILTIN_QOS_LIB_EXP
Builtin Qos Profiles, 689
DDS_BUILTIN_QOS_SNIPPET_LIB
Builtin Qos Profiles, 700
DDS_BUILTIN_TOPIC_KEY_UNKNOWN
Common types and functions, 508
DDS_BuiltinTopicKey_copy
Common types and functions, 504
DDS_BuiltinTopicKey_equals
Common types and functions, 504
DDS_BuiltinTopicKey_from_guid
Common types and functions, 505
DDS_BuiltinTopicKey_from_instance_handle
Common types and functions, 506
DDS_BuiltinTopicKey_t, 814
Common types and functions, 503
DDS_BuiltinTopicKey_to_guid
Common types and functions, 505
DDS_BuiltinTopicKey_to_instance_handle
Common types and functions, 505
DDS_BuiltinTopicReaderResourceLimits_t, 815
disable_fragmentation_support, 818
dynamically_allocate_fragmented_samples, 821
initial_fragmented_samples, 819
initial_infos, 817
initial_outstanding_reads, 817
initial_samples, 816
max_fragmented_samples, 819
max_fragmented_samples_per_remote_writer, 820
max_fragments_per_sample, 820
max_infos, 817
max_outstanding_reads, 818
max_samples, 816
max_samples_per_read, 818
DDS_BY_RECEPTION_TIMESTAMP_DESTINATIONORDER_QOS
DESTINATION_ORDER, 585
DDS_BY_SOURCE_TIMESTAMP_DESTINATIONORDER_QOS
DESTINATION_ORDER, 585
DDS_CdrPaddingKind
TYPESUPPORT, 652
DDS_ChannelSettings_t, 822
filter_expression, 822
multicast_settings, 822
priority, 823
DDS_ChannelSettingsSeq, 824
DDS_Char
DDS-Specific Primitive Types, 513
DDS_CharSeq, 824
DDS_CoherentSetInfo_copy
Data Samples, 154
DDS_CoherentSetInfo_equals
Data Samples, 153
DDS_CoherentSetInfo_t, 824
coherent_set_sequence_number, 825
group_coherent_set_sequence_number, 825
group_guid, 825
incomplete_coherent_set, 826

DDS_COMPRESSION_ID_BZIP2
Compression Settings, 573
DDS_COMPRESSION_ID_LZ4
Compression Settings, 574
DDS_COMPRESSION_ID_MASK_ALL
Compression Settings, 571
DDS_COMPRESSION_ID_MASK_NONE
Compression Settings, 571
DDS_COMPRESSION_ID_MASK_PUBLICATION_DEFAULT
Compression Settings, 572
DDS_COMPRESSION_ID_MASK_SUBSCRIPTION_DEFAULT
Compression Settings, 572
DDS_COMPRESSION_ID_ZLIB
Compression Settings, 573
DDS_COMPRESSION_LEVEL_BEST_COMPRESSION
Compression Settings, 571
DDS_COMPRESSION_LEVEL_BEST_SPEED
Compression Settings, 571
DDS_COMPRESSION_LEVEL_DEFAULT
Compression Settings, 572
DDS_COMPRESSION_THRESHOLD_DEFAULT
Compression Settings, 572
DDS_CompressionId_t
Compression Settings, 573
DDS_CompressionIdMask
Compression Settings, 573
DDS_CompressionSettings_t, 826
compression_ids, 826
writer_compression_level, 827
writer_compression_threshold, 827
DDS_ContentFilterProperty_t, 828
content_filter_topic_name, 829
expression_parameters, 829
filter_class_name, 829
filter_expression, 829
related_topic_name, 829
DDS_Cookie_t, 830
value, 830
DDS_CookieSeq, 830
DDS_DATA_AVAILABLE_STATUS
Status Kinds, 541
DDS_DATA_ON_READERS_STATUS
Status Kinds, 541
DDS_DATA_READER_CACHE_STATUS
Status Kinds, 546
DDS_DATA_READER_PROTOCOL_STATUS
Status Kinds, 546
DDS_DATA_REPRESENTATION_QOS_POLICY_ID
All QoS Policies, 558
DDS_DATA_REPRESENTATION_QOS_POLICY_NAME
DATA_REPRESENTATION, 569
DDS_DATA_WRITER_APPLICATION_ACKNOWLEDGMENT_STATUS
Status Kinds, 544
DDS_DATA_WRITER_CACHE_STATUS
Status Kinds, 545
DDS_DATA_WRITER_EMPTY_VIRTUAL_WRITER_STATUS
Status Kinds, 543
DDS_DATA_WRITER_INSTANCE_REPLACED_STATUS
Status Kinds, 544
DDS_DATA_WRITER_PROTOCOL_STATUS
Status Kinds, 545
DDS_DATA_WRITER_SAMPLE_REMOVED_STATUS
Status Kinds, 546
DDS_DATABASE_INTEGRATION_SERVICE_QOS
SERVICE, 637
DDS_DATABASE_QOS_POLICY_ID
All QoS Policies, 559
DDS_DATABASE_QOS_POLICY_NAME
DATABASE, 563
DDS_DatabaseQosPolicy, 831
cleanup_period, 833
initial_records, 833
initial_weak_references, 835
max_skiplist_level, 834
max_weak_references, 834
shutdown_cleanup_period, 833
shutdown_timeout, 832
thread, 832
DDS_DATAREADER_QOS_DEFAULT
Subscribers, 136
DDS_DATAREADER_QOS_USE_TOPIC_QOS
Subscribers, 137
DDS_DATAREADER_W_DATA_CONSISTENCY_CHECK
User Data Type Support, 75
DDS_DataReaderCacheStatus, 836
alive_instance_count, 840
alive_instance_count_peak, 840
compressed_sample_count, 842
content_filter_dropped_sample_count, 838
detached_instance_count, 842
detached_instance_count_peak, 842
disposed_instance_count, 841
disposed_instance_count_peak, 841
expired_dropped_sample_count, 839
no_writers_instance_count, 841
no_writers_instance_count_peak, 841
old_source_timestamp_dropped_sample_count, 837
ownership_dropped_sample_count, 838
replaced_dropped_sample_count, 839
sample_count, 837
sample_count_peak, 837
time_based_filter_dropped_sample_count, 838
tolerance_source_timestamp_dropped_sample_count, 838
total_samples_dropped_by_instance_replacement, 840
virtual_duplicate_dropped_sample_count, 839

writer_removed_batch_sample_dropped_sample_count, 840
DDS_DataReaderInstanceRemovalKind
DATA_READER_RESOURCE_LIMITS, 565
DDS_DATAREADERPROTOCOL_QOS_POLICY_ID
All QoS Policies, 559
DDS_DATAREADERPROTOCOL_QOS_POLICY_NAME
DATA_READER_PROTOCOL, 564
DDS_DataReaderProtocolQosPolicy, 843
disable_positive_acks, 845
expects_inline_qos, 844
propagate_dispose_of_unregistered_instances, 845
propagate_unregister_of_disposed_instances, 846
rtps_object_id, 844
rtps_reliable_reader, 846
virtual_guid, 844
DDS_DataReaderProtocolStatus, 847
dropped_fragment_count, 857
duplicate_sample_bytes, 851
duplicate_sample_bytes_change, 851
duplicate_sample_count, 851
duplicate_sample_count_change, 851
filtered_sample_bytes, 852
filtered_sample_bytes_change, 852
filtered_sample_count, 851
filtered_sample_count_change, 852
first_available_sample_sequence_number, 856
last_available_sample_sequence_number, 856
last_committed_sample_sequence_number, 856
out_of_range_rejected_sample_count, 856
reassembled_sample_count, 857
received_fragment_count, 857
received_gap_bytes, 855
received_gap_bytes_change, 855
received_gap_count, 854
received_gap_count_change, 855
received_heartbeat_bytes, 853
received_heartbeat_bytes_change, 853
received_heartbeat_count, 852
received_heartbeat_count_change, 852
received_sample_bytes, 850
received_sample_bytes_change, 850
received_sample_count, 849
received_sample_count_change, 850
rejected_sample_count, 855
rejected_sample_count_change, 855
sent_ack_bytes, 853
sent_ack_bytes_change, 854
sent_ack_count, 853
sent_ack_count_change, 853
sent_nack_bytes, 854
sent_nack_bytes_change, 854
sent_nack_count, 854
sent_nack_count_change, 854
sent_nack_fragment_bytes, 857
sent_nack_fragment_count, 857
uncommitted_sample_count, 856
DDS_DataReaderQos, 858
availability, 865
data_tags, 863
deadline, 861
destination_order, 862
durability, 861
history, 862
latency_budget, 861
liveliness, 862
multicast, 864
operator!=, 861
operator==, 860
ownership, 863
property, 865
protocol, 864
reader_data_lifecycle, 863
reader_resource_limits, 864
reliability, 862
representation, 863
resource_limits, 862
service, 865
subscription_name, 865
time_based_filter, 863
transport_priority, 865
transport_selection, 864
type_consistency, 863
type_support, 866
unicast, 864
user_data, 862
DDS_DataReaderQos_equals
DataReaders, 146
DDS_DATAREADERRESOURCELIMITS_QOS_POLICY_ID
All QoS Policies, 558
DDS_DATAREADERRESOURCELIMITS_QOS_POLICY_NAME
DATA_READER_RESOURCE_LIMITS, 567
DDS_DataReaderResourceLimitsInstanceReplacementSettings, 866
alive_instance_removal, 867
disposed_instance_removal, 867
no_writers_instance_removal, 867
DDS_DataReaderResourceLimitsQosPolicy, 867
autopurge_remote_not_alive_writer_delay, 884
disable_fragmentation_support, 874
dynamically_allocate_fragmented_samples, 876
initial_fragmented_samples, 875
initial_infos, 872
initial_outstanding_reads, 873
initial_remote_virtual_writers, 878
initial_remote_virtual_writers_per_instance, 879
initial_remote_writers, 871
initial_remote_writers_per_instance, 872

initial_topic_queries, 882
instance_replacement, 883
keep_minimum_state_for_instances, 881
lazily_allocate_batch_resources, 885
max_app_ack_response_length, 881
max_fragmented_samples, 874
max_fragmented_samples_per_remote_writer, 875
max_fragments_per_sample, 876
max_infos, 871
max_outstanding_reads, 873
max_query_condition_filters, 880
max_remote_virtual_writers, 878
max_remote_virtual_writers_per_instance, 879
max_remote_writers, 870
max_remote_writers_per_instance, 870
max_remote_writers_per_sample, 880
max_samples_per_read, 873
max_samples_per_remote_writer, 870
max_topic_queries, 882
max_total_instances, 877
shmem_ref_transfer_mode_attached_segment_allocation, 883
DDS_DataRepresentationId_t
DATA_REPRESENTATION, 568
DDS_DataRepresentationIdSeq, 886
DDS_DataRepresentationQosPolicy, 886
compression_settings, 888
value, 888
DDS_DATATAG_QOS_POLICY_ID
All QoS Policies, 558
DDS_DATATAG_QOS_POLICY_NAME
DATA_TAG, 579
DDS_DataTagQosPolicy
DATA_TAG, 575
DDS_DataTags, 889
tags, 889
DDS_DATAWRITER_CPP
User Data Type Support, 75
DDS_DATAWRITER_QOS_DEFAULT
Publishers, 114
DDS_DATAWRITER_QOS_USE_TOPIC_QOS
Publishers, 114
DDS_DataWriterCacheStatus, 889
alive_instance_count, 891
alive_instance_count_peak, 891
disposed_instance_count, 891
disposed_instance_count_peak, 891
sample_count, 890
sample_count_peak, 890
unregistered_instance_count, 891
unregistered_instance_count_peak, 892
DDS_DATAWRITERPROTOCOL_QOS_POLICY_ID
All QoS Policies, 559
DDS_DATAWRITERPROTOCOL_QOS_POLICY_NAME
DATA_WRITER_PROTOCOL, 579
DDS_DataWriterProtocolQosPolicy, 892
disable_inline_keyhash, 895
disable_positive_acks, 894
initial_virtual_sequence_number, 897
propagate_app_ack_with_no_response, 896
push_on_write, 894
rtps_object_id, 893
rtps_reliable_writer, 896
serialize_key_with_dispose, 895
virtual_guid, 893
DDS_DataWriterProtocolStatus, 897
filtered_sample_bytes, 901
filtered_sample_bytes_change, 901
filtered_sample_count, 901
filtered_sample_count_change, 901
first_available_sample_sequence_number, 906
first_available_sample_virtual_sequence_number, 906
first_unacknowledged_sample_sequence_number, 906
first_unacknowledged_sample_subscription_handle, 907
first_unacknowledged_sample_virtual_sequence_number, 907
first_unelapsed_keep_duration_sample_sequence_number, 907
last_available_sample_sequence_number, 906
last_available_sample_virtual_sequence_number, 906
pulled_fragment_bytes, 908
pulled_fragment_count, 908
pulled_sample_bytes, 903
pulled_sample_bytes_change, 903
pulled_sample_count, 902
pulled_sample_count_change, 902
pushed_fragment_bytes, 907
pushed_fragment_count, 907
pushed_sample_bytes, 900
pushed_sample_bytes_change, 900
pushed_sample_count, 900
pushed_sample_count_change, 900
received_ack_bytes, 903
received_ack_bytes_change, 904
received_ack_count, 903
received_ack_count_change, 903
received_nack_bytes, 904
received_nack_bytes_change, 904
received_nack_count, 904
received_nack_count_change, 904
received_nack_fragment_bytes, 908
received_nack_fragment_count, 908
rejected_sample_count, 905
rejected_sample_count_change, 905

send_window_size, 905
sent_gap_bytes, 905
sent_gap_bytes_change, 905
sent_gap_count, 904
sent_gap_count_change, 905
sent_heartbeat_bytes, 902
sent_heartbeat_bytes_change, 902
sent_heartbeat_count, 901
sent_heartbeat_count_change, 902
DDS_DataWriterQos, 909
availability, 916
batch, 916
data_tags, 915
deadline, 912
destination_order, 913
durability, 912
durability_service, 912
history, 913
latency_budget, 912
lifespan, 914
liveliness, 913
multi_channel, 916
operator!=, 911
operator==, 911
ownership, 914
ownership_strength, 914
property, 916
protocol, 915
publication_name, 916
publish_mode, 915
reliability, 913
representation, 914
resource_limits, 913
service, 916
topic_query_dispatch, 917
transfer_mode, 917
transport_priority, 913
transport_selection, 915
type_support, 917
unicast, 915
user_data, 914
writer_data_lifecycle, 914
writer_resource_limits, 915
DDS_DataWriterQos_equals
Data Writers, 117
DDS_DATAWRITERRESOURCELIMITS_QOS_POLICY_ID
All QoS Policies, 558
DDS_DATAWRITERRESOURCELIMITS_QOS_POLICY_NAME
DATA_WRITER_RESOURCE_LIMITS, 582
DDS_DataWriterResourceLimitsInstanceReplacementKind
DATA_WRITER_RESOURCE_LIMITS, 580
DDS_DataWriterResourceLimitsQosPolicy, 917
autoregister_instances, 922
initial_active_topic_queries, 924
initial_batches, 920
initial_concurrent_blocking_threads, 919
initial_remote_readers, 926
initial_virtual_writers, 923
initialize_writer_loaned_sample, 925
instance_replacement, 921
max_active_topic_queries, 924
max_app_ack_remote_readers, 924
max_batches, 921
max_concurrent_blocking_threads, 919
max_remote_reader_filters, 920
max_remote_readers, 923
max_virtual_writers, 923
replace_empty_instances, 922
writer_loaned_sample_allocation, 925
DDS_DataWriterShmemRefTransferModeSettings, 926
enable_data_consistency_check, 927
DDS_DATAWRITERTRANSFERMODE_QOS_POLICY_ID
All QoS Policies, 560
DDS_DATAWRITERTRANSFERMODE_QOS_POLICY_NAME
DATA_WRITER_TRANSFER_MODE, 583
DDS_DataWriterTransferModeQosPolicy, 927
shmem_ref_settings, 928
DDS_DEADLINE_QOS_POLICY_ID
All QoS Policies, 557
DDS_DEADLINE_QOS_POLICY_NAME
DEADLINE, 584
DDS_DeadlineQosPolicy, 928
period, 930
DDS_DEFAULT_FLOW_CONTROLLER_NAME
Flow Controllers, 125
DDS_DEFAULT_PRINT_FORMAT
Topics, 67
DDS_DELETE_PERSISTENT_JOURNAL
DURABILITY, 597
DDS_DESTINATIONORDER_QOS_POLICY_ID
All QoS Policies, 558
DDS_DESTINATIONORDER_QOS_POLICY_NAME
DESTINATION_ORDER, 586
DDS_DestinationOrderQosPolicy, 931
kind, 933
scope, 933
source_timestamp_tolerance, 933
DDS_DestinationOrderQosPolicyKind
DESTINATION_ORDER, 585
DDS_DestinationOrderQosPolicyScopeKind
DESTINATION_ORDER, 586
DDS_DISALLOW_TYPE_COERCION
TYPE_CONSISTENCY_ENFORCEMENT, 651
DDS_DISCOVERY_QOS_POLICY_ID
All QoS Policies, 558
DDS_DISCOVERY_QOS_POLICY_NAME
DISCOVERY, 587
DDS_DISCOVERY_SERVICE_SAMPLE

Sample Flags, 674
DDS_DISCOVERYCONFIG_BUILTIN_CHANNEL_MASK_ALL
DISCOVERY_CONFIG, 590
DDS_DISCOVERYCONFIG_BUILTIN_CHANNEL_MASK_DEFAULT
DISCOVERY_CONFIG, 589
DDS_DISCOVERYCONFIG_BUILTIN_CHANNEL_MASK_NONE
DISCOVERY_CONFIG, 589
DDS_DISCOVERYCONFIG_BUILTIN_DPSE
DISCOVERY_CONFIG, 592
DDS_DISCOVERYCONFIG_BUILTIN_PLUGIN_MASK_DEFAULT
DISCOVERY_CONFIG, 589
DDS_DISCOVERYCONFIG_BUILTIN_PLUGIN_MASK_NONE
DISCOVERY_CONFIG, 589
DDS_DISCOVERYCONFIG_BUILTIN_SDP
DISCOVERY_CONFIG, 591
DDS_DISCOVERYCONFIG_BUILTIN_SDP2
DISCOVERY_CONFIG, 592
DDS_DISCOVERYCONFIG_BUILTIN_SEDP
DISCOVERY_CONFIG, 591
DDS_DISCOVERYCONFIG_BUILTIN_SPDP
DISCOVERY_CONFIG, 591
DDS_DISCOVERYCONFIG_BUILTIN_SPDP2
DISCOVERY_CONFIG, 592
DDS_DISCOVERYCONFIG_QOS_POLICY_ID
All QoS Policies, 559
DDS_DISCOVERYCONFIG_QOS_POLICY_NAME
DISCOVERY_CONFIG, 594
DDS_DISCOVERYCONFIG_SERVICE_REQUEST_CHANNEL
DISCOVERY_CONFIG, 592
DDS_DISCOVERYCONFIG_TYPE_LOOKUP_SERVICE_CHANNEL
DISCOVERY_CONFIG, 592
DDS_DiscoveryConfigBuiltinChannelKind
DISCOVERY_CONFIG, 592
DDS_DiscoveryConfigBuiltinChannelKindMask
DISCOVERY_CONFIG, 590
DDS_DiscoveryConfigBuiltinPluginKind
DISCOVERY_CONFIG, 591
DDS_DiscoveryConfigBuiltinPluginKindMask
DISCOVERY_CONFIG, 590
DDS_DiscoveryConfigQosPolicy, 934
asynchronous_publisher, 949
builtin_discovery_plugins, 946
default_domain_announcement_period, 949
dns_tracker_polling_period, 956
enabled_builtin_channels, 946
endpoint_type_object_lb_serialization_threshold, 956
ignore_default_domain_announcements, 950
initial_participant_announcements, 939
locator_reachability_assert_period, 952
locator_reachability_change_detection_period, 953
locator_reachability_lease_duration, 953
max_initial_participant_announcement_period, 940
max_liveliness_loss_detection_period, 938
min_initial_participant_announcement_period, 940
new_remote_participant_announcements, 939
participant_announcement_period, 937
participant_configuration_reader, 959
participant_configuration_reader_resource_limits, 959
participant_configuration_writer, 957
participant_configuration_writer_data_lifecycle, 958
participant_configuration_writer_publish_mode, 957
participant_liveliness_assert_period, 937
participant_liveliness_lease_duration, 937
participant_message_reader, 947
participant_message_reader_reliability_kind, 946
participant_message_writer, 947
participant_reader_resource_limits, 941
publication_reader, 941
publication_reader_resource_limits, 942
publication_writer, 943
publication_writer_data_lifecycle, 944
publication_writer_publish_mode, 948
remote_participant_purge_kind, 938
request_types_filter, 959
secure_volatile_reader, 955
secure_volatile_writer, 954
secure_volatile_writer_publish_mode, 955
service_request_reader, 952
service_request_writer, 950
service_request_writer_data_lifecycle, 951
service_request_writer_publish_mode, 951
subscription_reader, 942
subscription_reader_resource_limits, 943
subscription_writer, 945
subscription_writer_data_lifecycle, 945
subscription_writer_publish_mode, 949
DDS_DiscoveryQosPolicy, 960
accept_unknown_peers, 963
enable_endpoint_discovery, 964
enabled_transports, 961
initial_peers, 962
metatraffic_transport_priority, 963
multicast_receive_addresses, 962
DDS_DISPOSED_INSTANCE_REPLACEMENT
DATA_WRITER_RESOURCE_LIMITS, 582
DDS_DISPOSED_THEN_ALIVE_INSTANCE_REPLACEMENT
DATA_WRITER_RESOURCE_LIMITS, 582
DDS_DOMAIN_ID_USE_XML_CONFIG
DomainParticipantConfigParams, 717
DDS_DomainId_t
DomainParticipants, 53
DDS_DomainParticipantConfigParams_t, 964
domain_entity_qos_library_name, 966
domain_entity_qos_profile_name, 966
domain_id, 965
participant_name, 965

participant_qos_library_name, 965
participant_qos_profile_name, 966
DDS_DomainParticipantConfigParams_t_INITIALIZER
DomainParticipantConfigParams, 717
DDS_DomainParticipantFactoryQos, 967
entity_factory, 969
logging, 969
monitoring, 969
operator!=, 968
operator==, 968
profile, 969
resource_limits, 969
DDS_DomainParticipantFactoryQos_equals
DomainParticipantFactory, 44
DDS_DomainParticipantProtocolStatus, 970
corrupted_rtps_message_count, 970
corrupted_rtps_message_count_change, 970
last_corrupted_message_timestamp, 970
DDS_DomainParticipantQos, 971
database, 975
default_unicast, 974
discovery, 974
discovery_config, 975
entity_factory, 974
event, 975
multicast_mapping, 976
operator!=, 973
operator==, 973
participant_name, 976
partition, 976
property, 975
receiver_pool, 975
resource_limits, 974
service, 976
transport_builtin, 974
type_support, 976
user_data, 973
wire_protocol, 974
DDS_DomainParticipantQos_equals
DomainParticipants, 53
DDS_DOMAINPARTICIPANTRESOURCELIMITS_QOS_POLICY_ID
All QoS Policies, 559
DDS_DOMAINPARTICIPANTRESOURCELIMITS_QOS_POLICY_NAME
DOMAIN_PARTICIPANT_RESOURCE_LIMITS, 596
DDS_DomainParticipantResourceLimitsIgnoredEntityReplacementKind
DOMAIN_PARTICIPANT_RESOURCE_LIMITS, 595
DDS_DomainParticipantResourceLimitsQosPolicy, 977
channel_filter_expression_max_length, 998
channel_seq_max_length, 997
content_filter_allocation, 984
content_filter_hash_buckets, 990
content_filtered_topic_allocation, 984
content_filtered_topic_hash_buckets, 990
contentfilter_property_max_length, 997
deserialized_type_object_dynamic_allocation_threshold, 996
flow_controller_allocation, 986
flow_controller_hash_buckets, 990
ignored_entity_allocation, 984
ignored_entity_hash_buckets, 989
ignored_entity_replacement_kind, 1001
local_publisher_allocation, 981
local_publisher_hash_buckets, 987
local_reader_allocation, 981
local_reader_hash_buckets, 986
local_subscriber_allocation, 981
local_subscriber_hash_buckets, 987
local_topic_allocation, 982
local_topic_hash_buckets, 987
local_writer_allocation, 981
local_writer_hash_buckets, 986
matching_reader_writer_pair_allocation, 983
matching_reader_writer_pair_hash_buckets, 989
matching_writer_reader_pair_allocation, 983
matching_writer_reader_pair_hash_buckets, 989
max_endpoint_group_cumulative_characters, 1000
max_endpoint_groups, 1000
max_gather_destinations, 991
max_partition_cumulative_characters, 993
max_partitions, 993
outstanding_asynchronous_sample_allocation, 985
participant_property_list_max_length, 998
participant_property_string_max_length, 998
participant_user_data_max_length, 991
publisher_group_data_max_length, 992
query_condition_allocation, 985
read_condition_allocation, 985
reader_data_tag_list_max_length, 1003
reader_data_tag_string_max_length, 1003
reader_property_list_max_length, 999
reader_property_string_max_length, 1000
reader_user_data_max_length, 993
remote_participant_allocation, 983
remote_participant_hash_buckets, 988
remote_reader_allocation, 982
remote_reader_hash_buckets, 988
remote_topic_query_allocation, 1001
remote_topic_query_hash_buckets, 1002
remote_writer_allocation, 982
remote_writer_hash_buckets, 988
serialized_type_object_dynamic_allocation_threshold, 995
shmem_ref_transfer_mode_max_segments, 1004
subscriber_group_data_max_length, 992
topic_data_max_length, 991
transport_info_list_max_length, 1001
type_code_max_serialized_length, 994
type_object_max_deserialized_length, 996

type_object_max_serialized_length, 994
writer_data_tag_list_max_length, 1002
writer_data_tag_string_max_length, 1003
writer_property_list_max_length, 999
writer_property_string_max_length, 999
writer_user_data_max_length, 992
DDS_Double
DDS-Specific Primitive Types, 515
DDS_DoubleSeq, 1004
DDS_DURABILITY_QOS_POLICY_ID
All QoS Policies, 557
DDS_DURABILITY_QOS_POLICY_NAME
DURABILITY, 599
DDS_DurabilityQosPolicy, 1005
direct_communication, 1007
kind, 1007
storage_settings, 1008
writer_depth, 1008
DDS_DurabilityQosPolicyKind
DURABILITY, 598
DDS_DURABILITYSERVICE_QOS_POLICY_ID
All QoS Policies, 558
DDS_DURABILITYSERVICE_QOS_POLICY_NAME
DURABILITY_SERVICE, 600
DDS_DurabilityServiceQosPolicy, 1009
history_depth, 1011
history_kind, 1010
max_instances, 1011
max_samples, 1011
max_samples_per_instance, 1012
service_cleanup_delay, 1010
DDS_DURATION_AUTO
Time Support, 523
DDS_DURATION_AUTO_NSEC
Time Support, 523
DDS_DURATION_AUTO_SEC
Time Support, 522
DDS_DURATION_INFINITE
Time Support, 522
DDS_DURATION_INFINITE_NSEC
Time Support, 522
DDS_DURATION_INFINITE_SEC
Time Support, 522
DDS_Duration_is_auto
Time Support, 521
DDS_Duration_is_infinite
Time Support, 520
DDS_Duration_is_zero
Time Support, 521
DDS_Duration_t, 1012
nanosec, 1013
sec, 1013
DDS_DURATION_ZERO
Time Support, 523
DDS_DURATION_ZERO_NSEC
Time Support, 523
DDS_DURATION_ZERO_SEC
Time Support, 523
DDS_DYNAMIC_DATA_JSON_PARSER_PROPERTIES_DEFAULT
Dynamic Data, 106
DDS_DYNAMIC_DATA_MEMBER_ID_UNSPECIFIED
Dynamic Data, 105
DDS_DYNAMIC_DATA_PROPERTY_DEFAULT
Dynamic Data, 105
DDS_DYNAMIC_DATA_TYPE_PROPERTY_DEFAULT
Dynamic Data, 106
DDS_DynamicData, 1013
~DDS_DynamicData, 1030
bind_complex_member, 1042
bind_type, 1041
clear_all_members, 1033
clear_member, 1034
clear_optional_member, 1033
copy, 1031
DDS_DynamicData, 1029
equal, 1032
from_cdr_buffer, 1037
from_string, 1039
get_boolean, 1056
get_boolean_array, 1073
get_boolean_seq, 1086
get_cdr_buffer, 1035
get_char, 1057
get_char_array, 1074
get_char_seq, 1087
get_complex_member, 1066
get_discriminator, 1067
get_double, 1056
get_double_array, 1072
get_double_seq, 1085
get_float, 1055
get_float_array, 1071
get_float_seq, 1084
get_info, 1040
get_int8, 1064
get_int8_array, 1079
get_int8_seq, 1092
get_long, 1051
get_long_array, 1067
get_long_seq, 1081
get_longdouble, 1060
get_longdouble_array, 1077
get_longdouble_seq, 1090
get_longlong, 1059
get_longlong_array, 1075
get_longlong_seq, 1088
get_member_count, 1045
get_member_info, 1047

get_member_info_by_index, 1049
get_member_type, 1050
get_octet, 1058
get_octet_array, 1075
get_octet_seq, 1087
get_short, 1052
get_short_array, 1068
get_short_seq, 1082
get_string, 1062
get_type, 1045
get_type_kind, 1045
get_uint8, 1065
get_uint8_array, 1080
get_uint8_seq, 1092
get_ulong, 1053
get_ulong_array, 1069
get_ulong_seq, 1082
get_ulonglong, 1060
get_ulonglong_array, 1076
get_ulonglong_seq, 1089
get_ushort, 1054
get_ushort_array, 1070
get_ushort_seq, 1083
get_wchar, 1061
get_wchar_array, 1078
get_wchar_seq, 1091
get_wstring, 1063
is_cdr, 1035
is_member_key, 1051
is_valid, 1030
member_exists, 1046
member_exists_in_type, 1047
operator=, 1032
operator==, 1032
print, 1040
set_boolean, 1098
set_boolean_array, 1112
set_boolean_seq, 1125
set_cdr_buffer, 1036
set_char, 1098
set_char_array, 1113
set_char_seq, 1125
set_complex_member, 1105
set_discriminator, 1132
set_double, 1097
set_double_array, 1111
set_double_seq, 1124
set_float, 1096
set_float_array, 1110
set_float_seq, 1123
set_int8, 1104
set_int8_array, 1118
set_int8_seq, 1130
set_long, 1093
set_long_array, 1107
set_long_seq, 1120
set_longdouble, 1101
set_longdouble_array, 1116
set_longdouble_seq, 1129
set_longlong, 1100
set_longlong_array, 1115
set_longlong_seq, 1127
set_octet, 1099
set_octet_array, 1114
set_octet_seq, 1126
set_short, 1094
set_short_array, 1108
set_short_seq, 1121
set_string, 1103
set_uint8, 1105
set_uint8_array, 1119
set_uint8_seq, 1131
set_ulong, 1095
set_ulong_array, 1109
set_ulong_seq, 1121
set_ulonglong, 1100
set_ulonglong_array, 1115
set_ulonglong_seq, 1128
set_ushort, 1095
set_ushort_array, 1109
set_ushort_seq, 1122
set_wchar, 1102
set_wchar_array, 1117
set_wchar_seq, 1129
set_wstring, 1103
to_cdr_buffer, 1037
to_cdr_buffer_ex, 1038
to_string, 1038
unbind_complex_member, 1044
unbind_type, 1042
DDS_DynamicDataInfo, 1133
member_count, 1134
stored_size, 1134
DDS_DynamicDataJsonParserProperties_t, 1134
DDS_DynamicDataMemberId
Dynamic Data, 105
DDS_DynamicDataMemberInfo, 1135
element_count, 1136
element_kind, 1136
member_exists, 1136
member_id, 1135
member_kind, 1136
member_name, 1135
DDS_DynamicDataProperty_t, 1137
buffer_initial_size, 1137
buffer_max_size, 1137
DDS_DynamicDataSeq, 1138
DDS_DynamicDataTypeProperty_t, 1138

data, 1139
serialization, 1139
DDS_DynamicDataTypeSerializationProperty_t, 1139
max_size_serialized, 1140
min_size_serialized, 1140
skip_deserialization, 1141
trim_to_size, 1141
use_42e_compatible_alignment, 1140
DDS_EDF_FLOW_CONTROLLER_SCHED_POLICY
Flow Controllers, 124
DDS_EMPTY_INSTANCE_REMOVAL
DATA_READER_RESOURCE_LIMITS, 566
DDS_EndpointGroup_t, 1142
quorum_count, 1142
role_name, 1142
DDS_EndpointGroupSeq, 1142
DDS_EndpointTrustAlgorithmInfo, 1143
Built-in Topic's Trust Types, 102
interceptor, 1143
DDS_EndpointTrustAttributesMask
Built-in Topic's Trust Types, 100
DDS_EndpointTrustInterceptorAlgorithmInfo, 1144
Built-in Topic's Trust Types, 102
required_mask, 1144
supported_mask, 1144
DDS_EndpointTrustProtectionInfo, 1144
bitmask, 1145
Built-in Topic's Trust Types, 101
plugin_bitmask, 1145
DDS_ENTITY_NAME_USE_XML_CONFIG
DomainParticipantConfigParams, 717
DDS_ENTITYFACTORY_QOS_POLICY_ID
All QoS Policies, 558
DDS_ENTITYFACTORY_QOS_POLICY_NAME
ENTITY_FACTORY, 601
DDS_EntityFactoryQosPolicy, 1145
autoenable_created_entities, 1147
DDS_ENTITYNAME_QOS_POLICY_ID
All QoS Policies, 559
DDS_ENTITYNAME_QOS_POLICY_NAME
ENTITY_NAME, 602
DDS_EntityNameQosPolicy, 1147
name, 1148
role_name, 1148
DDS_Enum
DDS-Specific Primitive Types, 516
DDS_EnumMember, 1149
name, 1149
ordinal, 1150
DDS_EnumMemberSeq, 1150
DDS_EVENT_QOS_POLICY_ID
All QoS Policies, 559
DDS_EVENT_QOS_POLICY_NAME
EVENT, 603
DDS_EventQosPolicy, 1150
initial_count, 1151
max_count, 1152
thread, 1151
DDS_ExceptionCode_t
Exception Codes, 530
DDS_EXCLUSIVE_OWNERSHIP_QOS
OWNERSHIP, 613
DDS_ExpressionProperty, 1152
key_only_filter, 1153
writer_side_filter_optimization, 1153
DDS_ExtensibilityKind
Type Code Support, 89
DDS_EXTENSIBLE_EXTENSIBILITY
Type Code Support, 90
DDS_FilterSampleInfo, 1153
related_reader_guid, 1154
related_sample_identity, 1154
related_source_guid, 1154
sample_identity, 1155
source_guid, 1155
DDS_FILTERSAMPLEINFO_DEFAULT
Topics, 72
DDS_FINAL_EXTENSIBILITY
Type Code Support, 90
DDS_FIXED_RATE_FLOW_CONTROLLER_NAME
Flow Controllers, 126
DDS_Float
DDS-Specific Primitive Types, 515
DDS_FloatSeq, 1156
DDS_FLOW_CONTROLLER_PROPERTY_DEFAULT
DomainParticipants, 60
DDS_FlowControllerProperty_t, 1156
scheduling_policy, 1157
token_bucket, 1157
DDS_FlowControllerSchedulingPolicy
Flow Controllers, 123
DDS_FlowControllerTokenBucketProperty_t, 1157
bytes_per_token, 1159
max_tokens, 1158
period, 1159
tokens_added_per_period, 1158
tokens_leaked_per_period, 1159
DDS_FULL_PERSISTENT_SYNCHRONIZATION
DURABILITY, 598
DDS_FULLY_PROCESSED_INSTANCE_REMOVAL
DATA_READER_RESOURCE_LIMITS, 566
DDS_g_tc_boolean
Type Code Support, 92
DDS_g_tc_char
Type Code Support, 93
DDS_g_tc_double
Type Code Support, 92
DDS_g_tc_float

Type Code Support, 92
DDS_g_tc_int8
Type Code Support, 93
DDS_g_tc_long
Type Code Support, 91
DDS_g_tc_longdouble
Type Code Support, 95
DDS_g_tc_longlong
Type Code Support, 94
DDS_g_tc_null
Type Code Support, 90
DDS_g_tc_octet
Type Code Support, 93
DDS_g_tc_short
Type Code Support, 90
DDS_g_tc_uint8
Type Code Support, 94
DDS_g_tc_ulong
Type Code Support, 91
DDS_g_tc_ulonglong
Type Code Support, 94
DDS_g_tc_ushort
Type Code Support, 91
DDS_g_tc_wchar
Type Code Support, 95
DDS_GROUP_PRESENTATION_QOS
PRESENTATION, 616
DDS_GROUPDATA_QOS_POLICY_ID
All QoS Policies, 558
DDS_GROUPDATA_QOS_POLICY_NAME
GROUP_DATA, 605
DDS_GroupDataQosPolicy, 1160
value, 1161
DDS_GUID_AUTO
GUID Support, 527
DDS_GUID_compare
GUID Support, 526
DDS_GUID_copy
GUID Support, 526
DDS_GUID_equals
GUID Support, 525
DDS_GUID_t, 1161
GUID Support, 525
value, 1162
DDS_GUID_UNKNOWN
GUID Support, 527
DDS_GUID_ZERO
GUID Support, 527
DDS_HANDLE_NIL
User Data Type Support, 78
DDS_Heap_calloc
Heap Support in C, 678
DDS_Heap_free
Heap Support in C, 678
DDS_Heap_malloc
Heap Support in C, 678
DDS_HIGHEST_OFFERED_PRESENTATION_QOS
PRESENTATION, 616
DDS_HISTORY_QOS_POLICY_ID
All QoS Policies, 558
DDS_HISTORY_QOS_POLICY_NAME
HISTORY, 604
DDS_HistoryQosPolicy, 1162
depth, 1164
kind, 1164
DDS_HistoryQosPolicyKind
HISTORY, 603
DDS_HPF_FLOW_CONTROLLER_SCHED_POLICY
Flow Controllers, 125
DDS_IMMUTABLE_TYPECODE_SYSTEM_EXCEPTION_CODE
Exception Codes, 530
DDS_INCOMPLETE_SNAPSHOT_TOPIC_QUERY
Sample Flags, 674
DDS_INCONSISTENT_TOPIC_STATUS
Status Kinds, 539
DDS_InconsistentTopicStatus, 1165
total_count, 1166
total_count_change, 1166
DDS_INSTANCE_PRESENTATION_QOS
PRESENTATION, 616
DDS_INSTANCE_SCOPE_DESTINATIONORDER_QOS
DESTINATION_ORDER, 586
DDS_INSTANCE_STATE_SERVICE_REQUEST_ID
ServiceRequest Built-in Topic, 498
DDS_InstanceHandle_compare
User Data Type Support, 77
DDS_InstanceHandle_copy
User Data Type Support, 77
DDS_InstanceHandle_equals
User Data Type Support, 76
DDS_InstanceHandle_is_nil
User Data Type Support, 78
DDS_InstanceHandle_t
User Data Type Support, 76
DDS_InstanceHandleSeq, 1166
DDS_InstanceStateConsistencyKind
RELIABILITY, 634
DDS_InstanceStateKind
Instance States, 165
DDS_InstanceStateMask
Instance States, 165
DDS_Int8
DDS-Specific Primitive Types, 514
DDS_Int8Seq, 1167
DDS_INTERMEDIATE_REPLY_SEQUENCE_SAMPLE
Sample Flags, 673
DDS_INTERMEDIATE_TOPIC_QUERY_SAMPLE
Sample Flags, 674

DDS_INTEROPERABLE_RTPS_WELL_KNOWN_PORTS
WIRE_PROTOCOL, 659
DDS_INVALID_LOCAL_ADVANCE_NOTICE_STATUS
Status Kinds, 543
DDS_INVALID_QOS_POLICY_ID
All QoS Policies, 557
DDS_InvalidLocalAdvanceNoticeStatus, 1167
expiration_code, 1168
expiration_message, 1168
expiration_time, 1168
DDS_JSON_PRINT_FORMAT
Topics, 67
DDS_KEEP_ALL_HISTORY_QOS
HISTORY, 604
DDS_KEEP_LAST_HISTORY_QOS
HISTORY, 604
DDS_KeyedOctets, 1168
~DDS_KeyedOctets, 1170
DDS_KeyedOctets, 1169
key, 1170
KeyedOctets Built-in Type, 511
length, 1170
value, 1170
DDS_KeyedOctetsSeq, 1170
DDS_KeyedString, 1171
~DDS_KeyedString, 1172
DDS_KeyedString, 1171, 1172
key, 1172
KeyedString Built-in Type, 509
value, 1173
DDS_KeyedStringSeq, 1173
DDS_LAST_SHARED_READER_QUEUE_SAMPLE
Sample Flags, 674
DDS_LATENCYBUDGET_QOS_POLICY_ID
All QoS Policies, 557
DDS_LATENCYBUDGET_QOS_POLICY_NAME
LATENCY_BUDGET, 606
DDS_LatencyBudgetQosPolicy, 1173
duration, 1175
DDS_LENGTH_AUTO
RECEIVER_POOL, 631
DDS_LENGTH_UNLIMITED
RESOURCE_LIMITS, 636
DDS_LIFESPAN_QOS_POLICY_ID
All QoS Policies, 558
DDS_LIFESPAN_QOS_POLICY_NAME
LIFESPAN, 606
DDS_LifespanQosPolicy, 1175
duration, 1176
DDS_LIVE_STREAM
Stream Kinds, 167
DDS_LIVELINESS_BASED_REMOTE_PARTICIPANT_PURGE
DISCOVERY_CONFIG, 593
DDS_LIVELINESS_CHANGED_STATUS
Status Kinds, 542
DDS_LIVELINESS_LOST_STATUS
Status Kinds, 542
DDS_LIVELINESS_QOS_POLICY_ID
All QoS Policies, 558
DDS_LIVELINESS_QOS_POLICY_NAME
LIVELINESS, 608
DDS_LivelinessChangedStatus, 1176
alive_count, 1178
alive_count_change, 1178
last_publication_handle, 1178
not_alive_count, 1178
not_alive_count_change, 1178
DDS_LivelinessLostStatus, 1179
total_count, 1179
total_count_change, 1180
DDS_LivelinessQosPolicy, 1180
assertions_per_lease_duration, 1183
kind, 1182
lease_duration, 1182
DDS_LivelinessQosPolicyKind
LIVELINESS, 607
DDS_LOCATOR_ADDRESS_INVALID
Common types and functions, 507
DDS_LOCATOR_ADDRESS_LENGTH_MAX
Common types and functions, 501
DDS_LOCATOR_INVALID
Common types and functions, 506
DDS_LOCATOR_KIND_INVALID
Common types and functions, 506
DDS_LOCATOR_KIND_RESERVED
Common types and functions, 508
DDS_LOCATOR_KIND_SHMEM
Common types and functions, 507
DDS_LOCATOR_KIND_SHMEM_510
Common types and functions, 507
DDS_LOCATOR_KIND_UDPv4
Common types and functions, 507
DDS_LOCATOR_KIND_UDPv4_WAN
Common types and functions, 507
DDS_LOCATOR_KIND_UDPv6
Common types and functions, 507
DDS_LOCATOR_KIND_UDPv6_510
Common types and functions, 508
DDS_LOCATOR_PORT_INVALID
Common types and functions, 506
DDS_LOCATOR_REACHABILITY_SERVICE_REQUEST_ID
ServiceRequest Built-in Topic, 498
DDS_Locator_t, 1184
address, 1184
Common types and functions, 503
kind, 1184
port, 1184
DDS_LOCATORFILTER_QOS_POLICY_ID

All QoS Policies, 559
DDS_LOCATORFILTER_QOS_POLICY_NAME
LOCATORFILTER, 609
DDS_LocatorFilter_t, 1185
filter_expression, 1185
locators, 1185
DDS_LocatorFilterQosPolicy, 1186
filter_name, 1187
locator_filters, 1187
DDS_LocatorFilterSeq, 1187
DDS_LocatorSeq, 1188
DDS_LOGGING_QOS_POLICY_ID
All QoS Policies, 560
DDS_LoggingQosPolicy, 1188
category, 1189
max_bytes_per_file, 1190
max_files, 1191
output_file, 1189
output_file_suffix, 1190
print_format, 1189
verbosity, 1189
DDS_Long
DDS-Specific Primitive Types, 515
DDS_LongDouble
DDS-Specific Primitive Types, 516
DDS_LongDoubleSeq, 1192
DDS_LongLong
DDS-Specific Primitive Types, 515
DDS_LongLongSeq, 1192
DDS_LongSeq, 1192
DDS_LOST_BY_AVAILABILITY_WAITING_TIME
DataReaders, 142
DDS_LOST_BY_DECODE_FAILURE
DataReaders, 143
DDS_LOST_BY_DESERIALIZATION_FAILURE
DataReaders, 143
DDS_LOST_BY_INCOMPLETE_COHERENT_SET
DataReaders, 141
DDS_LOST_BY_INSTANCES_LIMIT
DataReaders, 140
DDS_LOST_BY_LARGE_COHERENT_SET
DataReaders, 141
DDS_LOST_BY_OUT_OF_MEMORY
DataReaders, 143
DDS_LOST_BY_REMOTE_WRITER_SAMPLES_PER_VIRTUAL_QUEUE_LIMIT
DataReaders, 143
DDS_LOST_BY_REMOTE_WRITERS_PER_INSTANCE_LIMIT
DataReaders, 140
DDS_LOST_BY_REMOTE_WRITERS_PER_SAMPLE_LIMIT
DataReaders, 142
DDS_LOST_BY_SAMPLES_LIMIT
DataReaders, 144
DDS_LOST_BY_SAMPLES_PER_INSTANCE_LIMIT
DataReaders, 143
DDS_LOST_BY_SAMPLES_PER_REMOTE_WRITER_LIMIT
DataReaders, 142
DDS_LOST_BY_UNKNOWN_INSTANCE
DataReaders, 143
DDS_LOST_BY_VIRTUAL_WRITERS_LIMIT
DataReaders, 142
DDS_LOST_BY_WRITER
DataReaders, 140
DDS_MANUAL_BY_PARTICIPANT_LIVELINESS_QOS
LIVELINESS, 608
DDS_MANUAL_BY_TOPIC_LIVELINESS_QOS
LIVELINESS, 608
DDS_MEMORY_PERSISTENT_JOURNAL
DURABILITY, 597
DDS_MONITORING_LIBRARY_COMMAND_SERVICE_REQUEST_ID
ServiceRequest Built-in Topic, 498
DDS_MONITORING_LIBRARY_REPLY_SERVICE_REQUEST_ID
ServiceRequest Built-in Topic, 498
DDS_MONITORING_QOS_POLICY_ID
All QoS Policies, 560
DDS_MONITORING_QOS_POLICY_NAME
MONITORING, 611
DDS_MonitoringDedicatedParticipantSettings, 1193
collector_initial_peers, 1194
domain_id, 1194
enable, 1193
participant_qos_profile_name, 1194
DDS_MonitoringDistributionSettings, 1195
dedicated_participant, 1196
event_settings, 1196
logging_settings, 1197
periodic_settings, 1197
publisher_qos_profile_name, 1196
DDS_MonitoringEventDistributionSettings, 1197
concurrency_level, 1198
datawriter_qos_profile_name, 1199
publication_period, 1200
thread, 1199
DDS_MonitoringLoggingDistributionSettings, 1200
concurrency_level, 1201
datawriter_qos_profile_name, 1202
max_historical_logs, 1202
publication_period, 1203
thread, 1203
DDS_MonitoringLoggingForwardingSettings, 1204
middleware_forwarding_level, 1204
security_event_forwarding_level, 1204
service_forwarding_level, 1205
user_forwarding_level, 1205
DDS_MonitoringMetricSelection, 1206
disabled_metrics_selection, 1207
enabled_metrics_selection, 1207
resource_selection, 1206
DDS_MonitoringMetricSelectionSeq, 1208

DDS_MonitoringPeriodicDistributionSettings, 1209
datawriter_qos_profile_name, 1209
polling_period, 1210
thread, 1209
DDS_MonitoringQosPolicy, 1210
application_name, 1211
distribution_settings, 1212
enable, 1211
telemetry_data, 1212
DDS_MonitoringTelemetryData, 1213
logs, 1213
metrics, 1213
DDS_MULTICHANNEL_QOS_POLICY_ID
All QoS Policies, 559
DDS_MULTICHANNEL_QOS_POLICY_NAME
MULTICHANNEL, 611
DDS_MultiChannelQosPolicy, 1214
channels, 1215
filter_name, 1215
DDS_MUTABLE_EXTENSIBILITY
Type Code Support, 90
DDS_NEW_VIEW_STATE
View States, 164
DDS_NO_EXCEPTION_CODE
Exception Codes, 530
DDS_NO_INSTANCE_REMOVAL
DATA_READER_RESOURCE_LIMITS, 566
DDS_NO_MEMORY_SYSTEM_EXCEPTION_CODE
Exception Codes, 530
DDS_NO_RECOVER_INSTANCE_STATE_CONSISTENCY
RELIABILITY, 634
DDS_NO_REMOTE_PARTICIPANT_PURGE
DISCOVERY_CONFIG, 594
DDS_NO_REPLACEMENT_IGNORED_ENTITY_REPLACEMENT
DOMAIN_PARTICIPANT_RESOURCE_LIMITS, 595
DDS_NO_SERVICE_QOS
SERVICE, 637
DDS_NORMAL_PERSISTENT_SYNCHRONIZATION
DURABILITY, 598
DDS_NOT_ALIVE_DISPOSED_INSTANCE_STATE
Instance States, 166
DDS_NOT_ALIVE_FIRST_IGNORED_ENTITY_REPLACEMENT
DOMAIN_PARTICIPANT_RESOURCE_LIMITS, 595
DDS_NOT_ALIVE_INSTANCE_STATE
Instance States, 166
DDS_NOT_ALIVE_NO_WRITERS_INSTANCE_STATE
Instance States, 166
DDS_NOT_LOST
DataReaders, 140
DDS_NOT_NEW_VIEW_STATE
View States, 164
DDS_NOT_READ_SAMPLE_STATE
Sample States, 162
DDS_NOT_REJECTED
DataReaders, 144
DDS_NOT_SET_CDR_PADDING
TYPESUPPORT, 652
DDS_OBSERVABILITY_COLLECTOR_SERVICE_QOS
SERVICE, 637
DDS_Octet
DDS-Specific Primitive Types, 514
DDS_OctetBuffer_alloc
Octet Buffer Support, 755
DDS_OctetBuffer_dup
Octet Buffer Support, 755
DDS_OctetBuffer_free
Octet Buffer Support, 756
DDS_Octets, 1217
~DDS_Octets, 1218
DDS_Octets, 1217
length, 1218
Octets Built-in Type, 510
value, 1218
DDS_OctetSeq, 1218
DDS_OctetsSeq, 1219
DDS_OFF_PERSISTENT_JOURNAL
DURABILITY, 597
DDS_OFF_PERSISTENT_SYNCHRONIZATION
DURABILITY, 598
DDS_OFFERED_DEADLINE_MISSED_STATUS
Status Kinds, 539
DDS_OFFERED_INCOMPATIBLE_QOS_STATUS
Status Kinds, 540
DDS_OfferedDeadlineMissedStatus, 1219
last_instance_handle, 1220
total_count, 1220
total_count_change, 1220
DDS_OfferedIncompatibleQosStatus, 1221
last_policy_id, 1222
policies, 1222
total_count, 1221
total_count_change, 1221
DDS_ON_DEMAND_FLOW_CONTROLLER_NAME
Flow Controllers, 127
DDS_OWNERSHIP_QOS_POLICY_ID
All QoS Policies, 557
DDS_OWNERSHIP_QOS_POLICY_NAME
OWNERSHIP, 613
DDS_OwnershipQosPolicy, 1222
kind, 1227
DDS_OwnershipQosPolicyKind
OWNERSHIP, 612
DDS_OWNERSHIPSTRENGTH_QOS_POLICY_ID
All QoS Policies, 558
DDS_OWNERSHIPSTRENGTH_QOS_POLICY_NAME
OWNERSHIP_STRENGTH, 614
DDS_OwnershipStrengthQosPolicy, 1227
value, 1228

DDS_PARTICIPANT_CONFIG_PARAMS_DEFAULT
DomainParticipantFactory, 50
DDS_PARTICIPANT_QOS_DEFAULT
DomainParticipantFactory, 50
DDS_PARTICIPANT_TOPIC_NAME
Participant Built-in Topics, 491
DDS_ParticipantBuiltinTopicData, 1229
available_builtin_endpoints_ext, 1231
dds_builtin_endpoints, 1231
default_unicast_locators, 1232
domain_id, 1232
key, 1230
partial_configuration, 1234
Participant Built-in Topics, 491
participant_name, 1232
partition, 1233
product_version, 1232
property, 1230
reachability_lease_duration, 1233
rtps_protocol_version, 1230
rtps_vendor_id, 1230
transport_info, 1232
trust_algorithm_info, 1233
trust_protection_info, 1233
user_data, 1230
DDS_ParticipantBuiltinTopicDataSeq, 1235
DDS_ParticipantTrustAlgorithmInfo, 1235
Built-in Topic's Trust Types, 102
interceptor, 1236
key_establishment, 1236
signature, 1235
DDS_ParticipantTrustAttributesMask
Built-in Topic's Trust Types, 100
DDS_ParticipantTrustInterceptorAlgorithmInfo, 1236
Built-in Topic's Trust Types, 102
builtin_endpoints_required_mask, 1237
builtin_kx_endpoints_required_mask, 1237
supported_mask, 1237
user_endpoints_default_required_mask, 1237
DDS_ParticipantTrustKeyEstablishmentAlgorithmInfo, 1237
Built-in Topic's Trust Types, 102
shared_secret, 1238
DDS_ParticipantTrustProtectionInfo, 1238
bitmask, 1238
Built-in Topic's Trust Types, 100
plugin_bitmask, 1239
DDS_ParticipantTrustSignatureAlgorithmInfo, 1239
Built-in Topic's Trust Types, 101
message_auth, 1240
trust_chain, 1239
DDS_PARTITION_QOS_POLICY_ID
All QoS Policies, 558
DDS_PARTITION_QOS_POLICY_NAME
PARTITION, 614
DDS_PartitionQosPolicy, 1240
name, 1242
DDS_PERSIST_PERSISTENT_JOURNAL
DURABILITY, 597
DDS_PERSISTENCE_SERVICE_QOS
SERVICE, 637
DDS_PERSISTENT_DURABILITY_QOS
DURABILITY, 599
DDS_PersistentJournalKind
DURABILITY, 597
DDS_PersistentStorageSettings, 1242
enable, 1244
file_name, 1244
journal_kind, 1245
reader_checkpoint_frequency, 1248
restore, 1246
synchronization_kind, 1245
trace_file_name, 1244
vacuum, 1245
writer_instance_cache_allocation, 1246
writer_memory_state, 1248
writer_sample_cache_allocation, 1247
DDS_PersistentSynchronizationKind
DURABILITY, 597
DDS_PluginEndpointTrustAttributesMask
Built-in Topic's Trust Types, 101
DDS_PluginParticipantTrustAttributesMask
Built-in Topic's Trust Types, 100
DDS_PRESENTATION_QOS_POLICY_ID
All QoS Policies, 557
DDS_PRESENTATION_QOS_POLICY_NAME
PRESENTATION, 616
DDS_PresentationQosPolicy, 1249
access_scope, 1252
coherent_access, 1252
drop_incomplete_coherent_set, 1253
ordered_access, 1252
DDS_PresentationQosPolicyAccessScopeKind
PRESENTATION, 615
DDS_PRINT_FORMAT_PROPERTY_DEFAULT
Topics, 72
DDS_PrintFormatKind
Topics, 66
DDS_PrintFormatProperty, 1253
enum_as_int, 1254
include_root_elements, 1255
kind, 1254
pretty_print, 1254
Topics, 66
DDS_PRIVATE_MEMBER
Type Code Support, 85
DDS_ProductVersion_t, 1255
major, 1256

minor, 1256
release, 1256
revision, 1256
DDS_PRODUCTVERSION_UNKNOWN
Common types and functions, 503
DDS_PROFILE_BASELINE
Builtin Qos Profiles, 685
DDS_PROFILE_BASELINE_5_0_0
Builtin Qos Profiles, 685
DDS_PROFILE_BASELINE_5_1_0
Builtin Qos Profiles, 685
DDS_PROFILE_BASELINE_5_2_0
Builtin Qos Profiles, 686
DDS_PROFILE_BASELINE_5_3_0
Builtin Qos Profiles, 686
DDS_PROFILE_BASELINE_6_0_0
Builtin Qos Profiles, 686
DDS_PROFILE_BASELINE_6_1_0
Builtin Qos Profiles, 686
DDS_PROFILE_BASELINE_7_0_0
Builtin Qos Profiles, 686
DDS_PROFILE_BASELINE_7_1_0
Builtin Qos Profiles, 687
DDS_PROFILE_BASELINE_ROOT
Builtin Qos Profiles, 685
DDS_PROFILE_GENERIC_510_TRANSPORT_COMPATIBILITY
Builtin Qos Profiles, 688
DDS_PROFILE_GENERIC_AUTO_TUNING
Builtin Qos Profiles, 694
DDS_PROFILE_GENERIC_BEST_EFFORT
Builtin Qos Profiles, 689
DDS_PROFILE_GENERIC_COMMON
Builtin Qos Profiles, 687
DDS_PROFILE_GENERIC_CONNEXT_MICRO_COMPATIBILITY
Builtin Qos Profiles, 687
DDS_PROFILE_GENERIC_CONNEXT_MICRO_COMPATIBILITY_2_4_3
Builtin Qos Profiles, 688
DDS_PROFILE_GENERIC_CONNEXT_MICRO_COMPATIBILITY_2_4_9
Builtin Qos Profiles, 688
DDS_PROFILE_GENERIC_KEEP_LAST_RELIABLE
Builtin Qos Profiles, 689
DDS_PROFILE_GENERIC_KEEP_LAST_RELIABLE_LARGE_DATA
Builtin Qos Profiles, 692
DDS_PROFILE_GENERIC_KEEP_LAST_RELIABLE_LARGE_DATA_FAST_FLOW
Builtin Qos Profiles, 693
DDS_PROFILE_GENERIC_KEEP_LAST_RELIABLE_LARGE_DATA_MEDIUM_FLOW
Builtin Qos Profiles, 693
DDS_PROFILE_GENERIC_KEEP_LAST_RELIABLE_LARGE_DATA_SLOW_FLOW
Builtin Qos Profiles, 693
DDS_PROFILE_GENERIC_KEEP_LAST_RELIABLE_PERSISTENT
Builtin Qos Profiles, 694
DDS_PROFILE_GENERIC_KEEP_LAST_RELIABLE_TRANSIENT
Builtin Qos Profiles, 694
DDS_PROFILE_GENERIC_KEEP_LAST_RELIABLE_TRANSIENT_LOCAL
Builtin Qos Profiles, 694
DDS_PROFILE_GENERIC_MINIMAL_MEMORY_FOOTPRINT
Builtin Qos Profiles, 695
DDS_PROFILE_GENERIC_MONITORING2
Builtin Qos Profiles, 695
DDS_PROFILE_GENERIC_MONITORING2_WAN
Builtin Qos Profiles, 696
DDS_PROFILE_GENERIC_MONITORING_COMMON
Builtin Qos Profiles, 687
DDS_PROFILE_GENERIC_OTHER_DDS_VENDOR_COMPATIBILITY
Builtin Qos Profiles, 688
DDS_PROFILE_GENERIC_PARTICIPANT_LARGE_DATA
Builtin Qos Profiles, 690
DDS_PROFILE_GENERIC_PARTICIPANT_LARGE_DATA_MONITORING
Builtin Qos Profiles, 691
DDS_PROFILE_GENERIC_SECURITY
Builtin Qos Profiles, 695
DDS_PROFILE_GENERIC_STRICT_RELIABLE
Builtin Qos Profiles, 689
DDS_PROFILE_GENERIC_STRICT_RELIABLE_HIGH_THROUGHPUT
Builtin Qos Profiles, 690
DDS_PROFILE_GENERIC_STRICT_RELIABLE_LARGE_DATA
Builtin Qos Profiles, 691
DDS_PROFILE_GENERIC_STRICT_RELIABLE_LARGE_DATA_FAST_FLOW
Builtin Qos Profiles, 692
DDS_PROFILE_GENERIC_STRICT_RELIABLE_LARGE_DATA_MEDIUM_FLOW
Builtin Qos Profiles, 692
DDS_PROFILE_GENERIC_STRICT_RELIABLE_LARGE_DATA_SLOW_FLOW
Builtin Qos Profiles, 693
DDS_PROFILE_GENERIC_STRICT_RELIABLE_LOW_LATENCY
Builtin Qos Profiles, 690
DDS_PROFILE_PATTERN_ALARM_EVENT
Builtin Qos Profiles, 698
DDS_PROFILE_PATTERN_ALARM_STATUS
Builtin Qos Profiles, 699
DDS_PROFILE_PATTERN_EVENT
Builtin Qos Profiles, 698
DDS_PROFILE_PATTERN_LAST_VALUE_CACHE
Builtin Qos Profiles, 699
DDS_PROFILE_PATTERN_PERIODIC_DATA
Builtin Qos Profiles, 697
DDS_PROFILE_PATTERN_RELIABLE_STREAMING
Builtin Qos Profiles, 698
DDS_PROFILE_PATTERN_RPC
Builtin Qos Profiles, 697
DDS_PROFILE_PATTERN_STATUS
Builtin Qos Profiles, 699
DDS_PROFILE_PATTERN_STREAMING
Builtin Qos Profiles, 697
DDS_PROFILE_QOS_POLICY_ID
All QoS Policies, 559
DDS_PROFILE_QOS_POLICY_NAME
PROFILE, 617
DDS_ProtoQosPolicy, 1257

ignore_environment_profile, 1258
ignore_resource_profile, 1259
ignore_user_profile, 1258
string_profile, 1258
url_profile, 1258
DDS_PROPERTY_QOS_IMMUTABLE
PROPERTY, 620
DDS_PROPERTY_QOS_MUTABLE
PROPERTY, 620
DDS_PROPERTY_QOS_MUTABLE_UNTIL_ENABLE
PROPERTY, 620
DDS_PROPERTY_QOS_POLICY_ID
All QoS Policies, 559
DDS_PROPERTY_QOS_POLICY_NAME
PROPERTY, 626
DDS_Property_t, 1259
name, 1260
propagate, 1260
value, 1260
DDS_PropertyQosPolicy, 1260
PROPERTY, 619
value, 1262
DDS_PropertyQosPolicyHelper_get_property_mutability
PROPERTY, 621
DDS_PropertyQosPolicyMutability
PROPERTY, 620
DDS_PropertySeq, 1262
DDS_PROTOCOL_ACKNOWLEDGMENT_MODE
RELIABILITY, 633
DDS_PROTOCOLVERSION
Common types and functions, 502
DDS_PROTOCOLVERSION_1_0
Common types and functions, 501
DDS_PROTOCOLVERSION_1_1
Common types and functions, 502
DDS_PROTOCOLVERSION_1_2
Common types and functions, 502
DDS_PROTOCOLVERSION_2_0
Common types and functions, 502
DDS_PROTOCOLVERSION_2_1
Common types and functions, 502
DDS_ProtocolVersion_t, 1263
Common types and functions, 503
major, 1263
minor, 1263
DDS_PUBLIC_MEMBER
Type Code Support, 85
DDS_PUBLICATION_MATCHED_STATUS
Status Kinds, 543
DDS_PUBLICATION_PRIORITY_AUTOMATIC
PUBLISH_MODE, 628
DDS_PUBLICATION_PRIORITY_UNDEFINED
PUBLISH_MODE, 628
DDS_PUBLICATION_TOPIC_NAME
Publication Built-in Topics, 494
DDS_PublicationBuiltinTopicData, 1264
data_tags, 1269
deadline, 1267
destination_order, 1268
disable_positive_acks, 1272
durability, 1266
durability_service, 1267
group_data, 1269
key, 1266
latency_budget, 1267
lifespan, 1267
liveliness, 1267
locator_filter, 1271
ownership, 1268
ownership_strength, 1268
participant_key, 1266
partition, 1269
presentation, 1268
product_version, 1271
property, 1270
Publication Built-in Topics, 494
publication_name, 1272
publisher_key, 1270
related_datareader_key, 1273
reliability, 1267
representation, 1269
rtps_protocol_version, 1271
rtps_vendor_id, 1271
service, 1271
topic_data, 1269
topic_name, 1266
trust_algorithm_info, 1272
trust_protection_info, 1272
type_code, 1269
type_name, 1266
unicast_locators, 1270
user_data, 1268
virtual_guid, 1270
DDS_PublicationBuiltinTopicDataSeq, 1273
DDS_PublicationMatchedStatus, 1273
current_count, 1275
current_count_change, 1275
current_count_peak, 1275
last_subscription_handle, 1275
total_count, 1274
total_count_change, 1275
DDS_PUBLISHER_QOS_DEFAULT
DomainParticipants, 59
DDS_PublisherQos, 1276
asynchronous_publisher, 1279
entity_factory, 1278
group_data, 1278
operator!=, 1277

operator==, 1277
partition, 1278
presentation, 1278
publisher_name, 1279
DDS_PublisherQos_equals
Publishers, 108
DDS_PUBLISHMODE_QOS_POLICY_ID
All QoS Policies, 559
DDS_PUBLISHMODE_QOS_POLICY_NAME
PUBLISH_MODE, 629
DDS_PublishModeQosPolicy, 1279
flow_controller_name, 1281
kind, 1281
priority, 1282
DDS_PublishModeQosPolicyKind
PUBLISH_MODE, 628
DDS_QOS_ELEMENT_NAME_USE_XML_CONFIG
DomainParticipantConfigParams, 717
DDS_QOS_POLICY_COUNT
All QoS Policies, 557
DDS_QOS_PRINT_ALL
Infrastructure Module, 169
DDS_QosPolicyCount, 1283
count, 1284
policy_id, 1284
DDS_QosPolicyCountSeq, 1284
DDS_QosPolicyId_t
All QoS Policies, 557
DDS_QosPrintAll_t, 1285
DDS_QosPrintFormat, 1285
indent, 1286
is_standalone, 1285
print_private, 1286
DDS_QosPrintFormat_INITIALIZER
All QoS Policies, 556
DDS_QueryConditionParams, 1287
as_readconditionparams, 1288
query_expression, 1288
query_parameters, 1288
DDS_QUEUING_SERVICE_QOS
SERVICE, 637
DDS_READ_SAMPLE_STATE
Sample States, 162
DDS_ReadConditionParams, 1288
instance_states, 1289
sample_states, 1289
stream_kinds, 1289
view_states, 1289
DDS_READERDATALIFECYCLE_QOS_POLICY_ID
All QoS Policies, 558
DDS_READERDATALIFECYCLE_QOS_POLICY_NAME
READER_DATA_LIFECYCLE, 630
DDS_ReaderDataLifecycleQosPolicy, 1289
autopurge_disposed_instances_delay, 1291
autopurge_disposed_samples_delay, 1291
autopurge_nowriter_instances_delay, 1292
autopurge_nowriter_samples_delay, 1291
DDS_RECEIVERPOOL_QOS_POLICY_ID
All QoS Policies, 559
DDS_RECEIVERPOOL_QOS_POLICY_NAME
RECEIVER_POOL, 631
DDS_ReceiverPoolQosPolicy, 1293
buffer_alignment, 1295
buffer_size, 1294
thread, 1294
DDS_RECORDING_SERVICE_QOS
SERVICE, 637
DDS_RECOVER_INSTANCE_STATE_CONSISTENCY
RELIABILITY, 634
DDS_REDELIVERED_SAMPLE
Sample Flags, 673
DDS_REJECTED_BY_DECODE_FAILURE
DataReaders, 146
DDS_REJECTED_BY_INSTANCES_LIMIT
DataReaders, 144
DDS_REJECTED_BY_REMOTE_WRITER_SAMPLES_PER_VIRTUAL_QU
DataReaders, 145
DDS_REJECTED_BY_SAMPLES_LIMIT
DataReaders, 144
DDS_REJECTED_BY_SAMPLES_PER_INSTANCE_LIMIT
DataReaders, 145
DDS_REJECTED_BY_SAMPLES_PER_REMOTE_WRITER_LIMIT
DataReaders, 145
DDS_RELIABILITY_QOS_POLICY_ID
All QoS Policies, 558
DDS_RELIABILITY_QOS_POLICY_NAME
RELIABILITY, 635
DDS_ReliabilityQosPolicy, 1295
acknowledgment_kind, 1298
exact_reliability_kind_match, 1299
instance_state_consistency_kind, 1299
kind, 1298
max_blocking_time, 1298
DDS_ReliabilityQosPolicyAcknowledgmentModeKind
RELIABILITY, 633
DDS_ReliabilityQosPolicyKind
RELIABILITY, 632
DDS_RELIABLE_READER_ACTIVITY_CHANGED_STATUS
Status Kinds, 545
DDS_RELIABLE_RELIABILITY_QOS
RELIABILITY, 633
DDS_RELIABLE_WRITER_CACHE_CHANGED_STATUS
Status Kinds, 545
DDS_ReliableReaderActivityChangedStatus, 1300
active_count, 1301
active_count_change, 1301
inactive_count, 1301
inactive_count_change, 1302

last_instance_handle, 1302
DDS_ReliableWriterCacheChangedStatus, 1302
empty_reliable_writer_cache, 1303
full_reliable_writer_cache, 1303
high_watermark_reliable_writer_cache, 1304
low_watermark_reliable_writer_cache, 1303
replaced_unacknowledged_sample_count, 1304
unacknowledged_sample_count, 1304
unacknowledged_sample_count_peak, 1304
DDS_ReliableWriterCacheEventCount, 1305
total_count, 1305
total_count_change, 1305
DDS_RemoteParticipantPurgeKind
DISCOVERY_CONFIG, 593
DDS_REPLAY_SERVICE_QOS
SERVICE, 637
DDS_REPLICATE_SAMPLE
Sample Flags, 673
DDS_REQUESTED_DEADLINE_MISSED_STATUS
Status Kinds, 540
DDS_REQUESTED_INCOMPATIBLE_QOS_STATUS
Status Kinds, 540
DDS_RequestedDeadlineMissedStatus, 1306
last_instance_handle, 1306
total_count, 1306
total_count_change, 1306
DDS_RequestedIncompatibleQosStatus, 1307
last_policy_id, 1308
policies, 1308
total_count, 1307
total_count_change, 1308
DDS_RESOURCELIMITS_QOS_POLICY_ID
All QoS Policies, 558
DDS_RESOURCELIMITS_QOS_POLICY_NAME
RESOURCE_LIMITS, 635
DDS_ResourceLimitsQosPolicy, 1308
initial_instances, 1312
initial_samples, 1312
instance_hash_buckets, 1313
max_instances, 1311
max_samples, 1311
max_samples_per_instance, 1311
DDS_RETCODE_ALREADY_DELETED
Return Codes, 533
DDS_RETCODE_BAD_PARAMETER
Return Codes, 532
DDS_RETCODE_ERROR
Return Codes, 532
DDS_RETCODE_ILLEGAL_OPERATION
Return Codes, 533
DDS_RETCODE_IMMUTABLE_POLICY
Return Codes, 533
DDS_RETCODE_INCONSISTENT_POLICY
Return Codes, 533
DDS_RETCODE_NO_DATA
Return Codes, 533
DDS_RETCODE_NOT_ALLOWED_BY_SECURITY
Return Codes, 533
DDS_RETCODE_NOT_ENABLED
Return Codes, 533
DDS_RETCODE_OK
Return Codes, 532
DDS_RETCODE_OUT_OF_RESOURCES
Return Codes, 533
DDS_RETCODE_PRECONDITION_NOT_MET
Return Codes, 532
DDS_RETCODE_TIMEOUT
Return Codes, 533
DDS_RETCODE_UNSUPPORTED
Return Codes, 532
DDS_ReturnCode_t
Return Codes, 532
DDS_ROUTING_SERVICE_QOS
SERVICE, 637
DDS_RR_FLOW_CONTROLLER_SCHED_POLICY
Flow Controllers, 124
DDS_RTI_BACKWARDS_COMPATIBLE_RTPS_WELL_KNOWN_PORTS
WIRE_PROTOCOL, 658
DDS_RTPS_AUTO_ID_FROM_IP
WIRE_PROTOCOL, 658
DDS_RTPS_AUTO_ID_FROM_MAC
WIRE_PROTOCOL, 658
DDS_RTPS_AUTO_ID_FROM_UUID
WIRE_PROTOCOL, 658
DDS_RTPS_EntityId_t, 1313
GUID Support, 525
DDS_RTPS_GUID_t, 1314
GUID Support, 525
DDS_RTPS_GuidPrefix_t
GUID Support, 525
DDS_RTPS_RESERVED_PORT_BUILTIN_MULTICAST
WIRE_PROTOCOL, 658
DDS_RTPS_RESERVED_PORT_BUILTIN_UNICAST
WIRE_PROTOCOL, 658
DDS_RTPS_RESERVED_PORT_MASK_ALL
WIRE_PROTOCOL, 656
DDS_RTPS_RESERVED_PORT_MASK_DEFAULT
WIRE_PROTOCOL, 655
DDS_RTPS_RESERVED_PORT_MASK_NONE
WIRE_PROTOCOL, 656
DDS_RTPS_RESERVED_PORT_USER_MULTICAST
WIRE_PROTOCOL, 658
DDS_RTPS_RESERVED_PORT_USER_UNICAST
WIRE_PROTOCOL, 658
DDS_RtpsReliableReaderProtocol_t, 1314
app_ack_period, 1318
heartbeat_suppression_duration, 1316
max_heartbeat_response_delay, 1315

min_app_ack_response_keep_duration, 1318
min_heartbeat_response_delay, 1315
nack_period, 1316
receive_window_size, 1317
round_trip_time, 1317
samples_per_app_ack, 1319
DDS_RtpsReliableWriterProtocol_t, 1320
disable_positive_acks_decrease_sample_keep_duration_factor, 1332
disable_positive_acks_enable_adaptive_sample_keep_duration, 1331
disable_positive_acks_increase_sample_keep_duration_factor, 1332
disable_positive_acks_max_sample_keep_duration, 1331
disable_positive_acks_min_sample_keep_duration, 1330
disable_repair_piggyback_heartbeat, 1338
enable_multicast_periodic_heartbeat, 1337
fast_heartbeat_period, 1323
heartbeat_period, 1323
heartbeats_per_max_samples, 1327
high_watermark, 1322
inactivate_nonprogressing_readers, 1326
late_joiner_heartbeat_period, 1324
low_watermark, 1321
max_bytes_per_nack_response, 1329
max_heartbeat_retries, 1326
max_nack_response_delay, 1328
max_send_window_size, 1333
min_nack_response_delay, 1328
min_send_window_size, 1333
multicast_resend_threshold, 1337
nack_suppression_duration, 1329
samples_per_virtual_heartbeat, 1325
send_window_decrease_factor, 1336
send_window_increase_factor, 1335
send_window_update_period, 1334
virtual_heartbeat_period, 1325
DDS_RtpsReservedPortKind
WIRE_PROTOCOL, 657
DDS_RtpsReservedPortKindMask
WIRE_PROTOCOL, 657
DDS_RtpsWellKnownPorts_t, 1338
builtin_multicast_port_offset, 1342
builtin_unicast_port_offset, 1342
domain_id_gain, 1341
participant_id_gain, 1342
port_base, 1340
user_multicast_port_offset, 1343
user_unicast_port_offset, 1343
DDS_SAMPLE_LOST_STATUS
Status Kinds, 541
DDS_SAMPLE_REJECTED_STATUS
Status Kinds, 541
DDS_SampleFlag
Sample Flags, 673
DDS_SampleFlagBits
Sample Flags, 673
DDS_SampleIdentity_equals
WriteParams, 675
DDS_SampleIdentity_t, 1343
DDS_SampleInfo, 1344
absolute_generation_rank, 1350
coherent_set_info, 1354
disposed_generation_count, 1349
flag, 1353
generation_rank, 1350
instance_handle, 1349
instance_state, 1348
no_writers_generation_count, 1349
original_publication_virtual_guid, 1352
original_publication_virtual_sequence_number, 1352
publication_handle, 1349
publication_sequence_number, 1351
reception_sequence_number, 1352
reception_timestamp, 1351
related_original_publication_virtual_guid, 1352
related_original_publication_virtual_sequence_number, 1353
related_source_guid, 1353
related_subscription_guid, 1354
sample_rank, 1350
sample_state, 1347
source_guid, 1353
source_timestamp, 1348
topic_query_guid, 1354
valid_data, 1351
view_state, 1348
DDS_SampleInfo_get_related_sample_identity
Data Samples, 154
DDS_SampleInfo_get_sample_identity
Data Samples, 154
DDS_SampleInfoSeq, 1355
DDS_SampleLostStatus, 1355
last_reason, 1356
total_count, 1355
total_count_change, 1355
DDS_SampleLostStatusKind
DataReaders, 140
DDS_SampleRejectedStatus, 1356
last_instance_handle, 1357
last_reason, 1357
total_count, 1356
total_count_change, 1357
DDS_SampleRejectedStatusKind
DataReaders, 144
DDS_SampleStateKind

Sample States, 161
DDS_SampleStateMask
Sample States, 161
DDS_SEQUENCE_NUMBER_MAX
Sequence Number Support, 529
DDS_SEQUENCE_NUMBER_UNKNOWN
Sequence Number Support, 528
DDS_SEQUENCE_NUMBER_ZERO
Sequence Number Support, 528
DDS_SequenceNumber_t, 1357
high, 1358
low, 1358
Sequence Number Support, 528
DDS_SERVICE_QOS_POLICY_NAME
SERVICE, 638
DDS_SERVICE_REQUEST_ACCEPTED_STATUS
Status Kinds, 544
DDS_SERVICE_REQUEST_TOPIC_NAME
ServiceRequest Built-in Topic, 499
DDS_ServiceQosPolicy, 1358
kind, 1359
DDS_ServiceQosPolicyKind
SERVICE, 637
DDS_ServiceRequest, 1359
instance_id, 1360
request_body, 1360
service_id, 1360
ServiceRequest Built-in Topic, 497
DDS_ServiceRequestAcceptedStatus, 1361
current_count, 1362
current_count_change, 1362
last_request_handle, 1362
service_id, 1362
total_count, 1362
total_count_change, 1362
DDS_ServiceRequestSeq, 1363
DDS_SHARED_OWNERSHIP_QOS
OWNERSHIP, 613
DDS_Short
DDS-Specific Primitive Types, 514
DDS_ShortSeq, 1363
DDS_SNIPPET_5_1_0_TRANSPORT_ENABLE
Builtin Qos Profiles, 716
DDS_SNIPPET_COMPATIBILITY_CONNEXT_MICRO_VERSION_2_4_3
Builtin Qos Profiles, 715
DDS_SNIPPET_COMPATIBILITY_OTHER_DDS_VENDORS_ENABLE
Builtin Qos Profiles, 715
DDS_SNIPPET_FEATURE_AUTO_TUNING_ENABLE
Builtin Qos Profiles, 710
DDS_SNIPPET_FEATURE_FLOW_CONTROLLER_209MBPS
Builtin Qos Profiles, 709
DDS_SNIPPET_FEATURE_FLOW_CONTROLLER_52MBPS
Builtin Qos Profiles, 709
DDS_SNIPPET_FEATURE_FLOW_CONTROLLER_838MBPS
Builtin Qos Profiles, 709
DDS_SNIPPET_FEATURE_MONITORING2_ENABLE
Builtin Qos Profiles, 711
DDS_SNIPPET_FEATURE_MONITORING_ENABLE
Builtin Qos Profiles, 710
DDS_SNIPPET_FEATURE_SECURITY_ENABLE
Builtin Qos Profiles, 711
DDS_SNIPPET_FEATURE_TOPIC_QUERY_ENABLE
Builtin Qos Profiles, 711
DDS_SNIPPET_FEATURE_TYPE_SUPPORT_V1_DISABLE
Builtin Qos Profiles, 712
DDS_SNIPPET_FEATURE_TYPE_SUPPORT_V1_ENABLE
Builtin Qos Profiles, 712
DDS_SNIPPET_FEATURE_TYPE_SUPPORT_V2_DISABLE
Builtin Qos Profiles, 713
DDS_SNIPPET_FEATURE_TYPE_SUPPORT_V2_ENABLE
Builtin Qos Profiles, 712
DDS_SNIPPET_OPTIMIZATION_DATACACHE_LARGE_DATA_DYNAMICM
Builtin Qos Profiles, 703
DDS_SNIPPET_OPTIMIZATION_DISCOVERY_COMMON
Builtin Qos Profiles, 703
DDS_SNIPPET_OPTIMIZATION_DISCOVERY_ENDPOINT_FAST
Builtin Qos Profiles, 704
DDS_SNIPPET_OPTIMIZATION_DISCOVERY_PARTICIPANT_COMPACT
Builtin Qos Profiles, 704
DDS_SNIPPET_OPTIMIZATION_RELIABILITY_PROTOCOL_COMMON
Builtin Qos Profiles, 700
DDS_SNIPPET_OPTIMIZATION_RELIABILITY_PROTOCOL_HIGH_RATE
Builtin Qos Profiles, 701
DDS_SNIPPET_OPTIMIZATION_RELIABILITY_PROTOCOL_KEEP_ALL
Builtin Qos Profiles, 700
DDS_SNIPPET_OPTIMIZATION_RELIABILITY_PROTOCOL_KEEP_LAST
Builtin Qos Profiles, 701
DDS_SNIPPET_OPTIMIZATION_RELIABILITY_PROTOCOL_LARGE_DATA
Builtin Qos Profiles, 702
DDS_SNIPPET_OPTIMIZATION_RELIABILITY_PROTOCOL_LOW_LATEN
Builtin Qos Profiles, 702
DDS_SNIPPET_OPTIMIZATION_TRANSPORT_LARGE_BUFFERS
Builtin Qos Profiles, 705
DDS_SNIPPET_QOS_POLICY_BATCHING_ENABLE
Builtin Qos Profiles, 708
DDS_SNIPPET_QOS_POLICY_DURABILITY_PERSISTENT
Builtin Qos Profiles, 708
DDS_SNIPPET_QOS_POLICY_DURABILITY_TRANSIENT
Builtin Qos Profiles, 707
DDS_SNIPPET_QOS_POLICY_DURABILITY_TRANSIENT_LOCAL
Builtin Qos Profiles, 707
DDS_SNIPPET_QOS_POLICY_HISTORY_KEEP_ALL
Builtin Qos Profiles, 706
DDS_SNIPPET_QOS_POLICY_HISTORY_KEEP_LAST_1
Builtin Qos Profiles, 706
DDS_SNIPPET_QOS_POLICY_PUBLISH_MODE_ASYNCHRONOUS
Builtin Qos Profiles, 706
DDS_SNIPPET_QOS_POLICY_RELIABILITY_BEST_EFFORT

Builtin Qos Profiles, 705
DDS_SNIPPET_QOS_POLICY_RELIABILITY_RELIABLE
Builtin Qos Profiles, 705
DDS_SNIPPET_TRANSPORT_TCP_LAN_CLIENT
Builtin Qos Profiles, 713
DDS_SNIPPET_TRANSPORT_TCP_WAN_ASYMMETRIC_CLIENT
Builtin Qos Profiles, 714
DDS_SNIPPET_TRANSPORT_TCP_WAN_ASYMMETRIC_SERVER
Builtin Qos Profiles, 714
DDS_SNIPPET_TRANSPORT_TCP_WAN_SYMMETRIC_CLIENT
Builtin Qos Profiles, 713
DDS_SNIPPET_TRANSPORT_UDP_AVOID_IP_FRAGMENTATION
Builtin Qos Profiles, 714
DDS_SNIPPET_TRANSPORT_UDP_WAN
Builtin Qos Profiles, 715
DDS_SQLFILTER_NAME
DomainParticipants, 61
DDS_STATUS_MASK_ALL
Status Kinds, 537
DDS_STATUS_MASK_NONE
Status Kinds, 537
DDS_StatusKind
Status Kinds, 538
DDS_StatusMask
Status Kinds, 537
DDS_StreamKind
Stream Kinds, 167
DDS_StreamKindMask
Stream Kinds, 167
DDS_String_alloc
String Support, 759
DDS_String_dup
String Support, 761
DDS_String_free
String Support, 762
DDS_String_replace
String Support, 761
DDS_STRINGMATCHFILTER_NAME
DomainParticipants, 61
DDS_StringSeq, 1364
DDS_StructMember, 1365
bits, 1366
id, 1366
is_key, 1366
is_must_understand, 1366
is_optional, 1366
is_pointer, 1366
name, 1365
type, 1365
DDS_StructMemberSeq, 1367
DDS_SUBSCRIBER_QOS_DEFAULT
DomainParticipants, 59
DDS_SubscriberQos, 1367
entity_factory, 1369
group_data, 1369
operator!=, 1368
operator==, 1368
partition, 1369
presentation, 1369
subscriber_name, 1370
DDS_SubscriberQos_equals
Subscribers, 131
DDS_SUBSCRIPTION_MATCHED_STATUS
Status Kinds, 543
DDS_SUBSCRIPTION_TOPIC_NAME
Subscription Built-in Topics, 496
DDS_SubscriptionBuiltinTopicData, 1370
content_filter_property, 1377
data_tags, 1375
deadline, 1373
destination_order, 1374
disable_positive_acks, 1378
durability, 1373
group_data, 1375
key, 1372
latency_budget, 1373
liveliness, 1373
multicast_locators, 1376
ownership, 1374
participant_key, 1372
partition, 1375
presentation, 1374
product_version, 1378
property, 1376
related_datawriter_key, 1379
reliability, 1373
representation, 1375
rtps_protocol_version, 1377
rtps_vendor_id, 1377
service, 1377
subscriber_key, 1376
Subscription Built-in Topics, 495
subscription_name, 1378
time_based_filter, 1374
topic_data, 1375
topic_name, 1372
trust_algorithm_info, 1378
trust_protection_info, 1378
type_code, 1376
type_consistency, 1375
type_name, 1372
unicast_locators, 1376
user_data, 1374
virtual_guid, 1377
DDS_SubscriptionBuiltinTopicDataSeq, 1379
DDS_SubscriptionMatchedStatus, 1379
current_count, 1381
current_count_change, 1381

current_count_peak, 1381
last_publication_handle, 1381
total_count, 1380
total_count_change, 1381
DDS_SYNCHRONOUS_PUBLISH_MODE_QOS
PUBLISH_MODE, 629
DDS_SYSTEM_EXCEPTION_CODE
Exception Codes, 530
DDS_SYSTEMRESOURCELIMITS_QOS_POLICY_ID
All QoS Policies, 559
DDS_SYSTEMRESOURCELIMITS_QOS_POLICY_NAME
SYSTEM_RESOURCE_LIMITS, 638
DDS_SystemResourceLimitsQosPolicy, 1382
initial_objects_per_thread, 1383
max_objects_per_thread, 1383
DDS_Tag, 1384
name, 1384
value, 1385
DDS_TagSeq, 1385
DDS_TCKind
Type Code Support, 89
DDS_THREAD_SETTINGS_CANCEL_ASYNCHRONOUS
Thread Settings, 548
DDS_THREAD_SETTINGS_CPU_NO_ROTATION
Thread Settings, 549
DDS_THREAD_SETTINGS_CPU_RR_ROTATION
Thread Settings, 549
DDS_THREAD_SETTINGS_FLOATING_POINT
Thread Settings, 548
DDS_THREAD_SETTINGS_KIND_MASK_DEFAULT
Thread Settings, 547
DDS_THREAD_SETTINGS_PRIORITY_ENFORCE
Thread Settings, 548
DDS_THREAD_SETTINGS_REALTIME_PRIORITY
Thread Settings, 548
DDS_THREAD_SETTINGS_STDIO
Thread Settings, 548
DDS_ThreadSettings_t, 1385
cpu_list, 1386
cpu_rotation, 1387
mask, 1386
priority, 1386
stack_size, 1386
DDS_ThreadSettingsCpuRotationKind
Thread Settings, 548
DDS_ThreadSettingsKind
Thread Settings, 548
DDS_ThreadSettingsKindMask
Thread Settings, 547
DDS_TIME_INVALID
Time Support, 522
DDS_TIME_INVALID_NSEC
Time Support, 522
DDS_TIME_INVALID_SEC
Time Support, 521
DDS_Time_is_invalid
Time Support, 520
DDS_Time_is_zero
Time Support, 520
DDS_TIME_MAX
Time Support, 521
DDS_Time_t, 1387
nanosec, 1388
sec, 1388
DDS_TIME_ZERO
Time Support, 518
DDS_TIMEBASEDFILTER_QOS_POLICY_ID
All QoS Policies, 558
DDS_TIMEBASEDFILTER_QOS_POLICY_NAME
TIME_BASED_FILTER, 639
DDS_TimeBasedFilterQosPolicy, 1389
minimum_separation, 1391
DDS_TK_ALIAS
Type Code Support, 89
DDS_TK_ARRAY
Type Code Support, 89
DDS_TK_BOOLEAN
Type Code Support, 89
DDS_TK_CHAR
Type Code Support, 89
DDS_TK_DOUBLE
Type Code Support, 89
DDS_TK_ENUM
Type Code Support, 89
DDS_TK_FLOAT
Type Code Support, 89
DDS_TK_INT8
Type Code Support, 89
DDS_TK_LONG
Type Code Support, 89
DDS_TK_LONGDOUBLE
Type Code Support, 89
DDS_TK_LONGLONG
Type Code Support, 89
DDS_TK_NULL
Type Code Support, 89
DDS_TK_OCTET
Type Code Support, 89
DDS_TK_SEQUENCE
Type Code Support, 89
DDS_TK_SHORT
Type Code Support, 89
DDS_TK_STRING
Type Code Support, 89
DDS_TK_STRUCT
Type Code Support, 89
DDS_TK_UINT8
Type Code Support, 89

DDS_TK_ULONG
Type Code Support, 89
DDS_TK_ULONGLONG
Type Code Support, 89
DDS_TK_UNION
Type Code Support, 89
DDS_TK_USHORT
Type Code Support, 89
DDS_TK_VALUE
Type Code Support, 89
DDS_TK_WCHAR
Type Code Support, 89
DDS_TK_WSTRING
Type Code Support, 89
DDS_TOPIC_PRESENTATION_QOS
PRESENTATION, 616
DDS_TOPIC_QOS_DEFAULT
DomainParticipants, 58
DDS_TOPIC_QUERY_SELECTION_KIND_CONTINUOUS
Topic Queries, 160
DDS_TOPIC_QUERY_SELECTION_KIND_HISTORY_SNAPSHOT
Topic Queries, 160
DDS_TOPIC_QUERY_SELECTION_SELECT_ALL
Topic Queries, 160
DDS_TOPIC_QUERY_SELECTION_USE_READER_CONTENT_FILTER
Topic Queries, 160
DDS_TOPIC_QUERY_SERVICE_REQUEST_ID
ServiceRequest Built-in Topic, 498
DDS_TOPIC_QUERY_STREAM
Stream Kinds, 167
DDS_TOPIC_SCOPE_DESTINATIONORDER_QOS
DESTINATION_ORDER, 586
DDS_TOPIC_TOPIC_NAME
Topic Built-in Topics, 493
DDS_TopicBuiltinTopicData, 1391
deadline, 1393
destination_order, 1394
durability, 1393
durability_service, 1393
history, 1395
key, 1392
latency_budget, 1394
lifespan, 1394
liveliness, 1394
name, 1392
ownership, 1395
reliability, 1394
representation, 1395
resource_limits, 1395
Topic Built-in Topics, 492
topic_data, 1395
transport_priority, 1394
type_name, 1393
DDS_TopicBuiltinTopicDataSeq, 1396
DDS_TOPICDATA_QOS_POLICY_ID
All QoS Policies, 558
DDS_TOPICDATA_QOS_POLICY_NAME
TOPIC_DATA, 640
DDS_TopicDataQosPolicy, 1396
value, 1397
DDS_TopicQos, 1397
deadline, 1400
destination_order, 1401
durability, 1400
durability_service, 1400
history, 1401
latency_budget, 1401
lifespan, 1402
liveliness, 1401
operator!=, 1399
operator==, 1399
ownership, 1402
reliability, 1401
representation, 1402
resource_limits, 1401
topic_data, 1400
transport_priority, 1402
DDS_TopicQos_equals
Topics, 167
DDS_TopicQueryData, 1402
original_related_reader_guid, 1403
Topic Queries, 159
topic_name, 1403
topic_query_selection, 1403
DDS_TOPICQUERYDISPATCH_QOS_POLICY_ID
All QoS Policies, 560
DDS_TOPICQUERYDISPATCH_QOS_POLICY_NAME
TOPIC_QUERY_DISPATCH, 641
DDS_TopicQueryDispatchQosPolicy, 1404
enable, 1405
publication_period, 1405
samples_per_period, 1405
DDS_TopicQuerySelection, 1406
filter_class_name, 1407
filter_expression, 1407
filter_parameters, 1407
kind, 1407
Topic Queries, 158
DDS_TopicQuerySelectionKind
Topic Queries, 159
DDS_TRANSIENT_DURABILITY_QOS
DURABILITY, 599
DDS_TRANSIENT_LOCAL_DURABILITY_QOS
DURABILITY, 599
DDS_TRANSPORTBUILTIN_MASK_ALL
TRANSPORT_BUILTIN, 643
DDS_TRANSPORTBUILTIN_MASK_DEFAULT
TRANSPORT_BUILTIN, 642

DDS_TRANSPORTBUILTIN_MASK_NONE
TRANSPORT_BUILTIN, 642
DDS_TRANSPORTBUILTIN_QOS_POLICY_ID
All QoS Policies, 559
DDS_TRANSPORTBUILTIN_QOS_POLICY_NAME
TRANSPORT_BUILTIN, 644
DDS_TRANSPORTBUILTIN_SHMEM
TRANSPORT_BUILTIN, 644
DDS_TRANSPORTBUILTIN_SHMEM_ALIAS
TRANSPORT_BUILTIN, 644
DDS_TRANSPORTBUILTIN_UDPv4
TRANSPORT_BUILTIN, 644
DDS_TRANSPORTBUILTIN_UDPv4_ALIAS
TRANSPORT_BUILTIN, 644
DDS_TRANSPORTBUILTIN_UDPv4_WAN
TRANSPORT_BUILTIN, 644
DDS_TRANSPORTBUILTIN_UDPv4_WAN_ALIAS
TRANSPORT_BUILTIN, 644
DDS_TRANSPORTBUILTIN_UDPv6
TRANSPORT_BUILTIN, 644
DDS_TRANSPORTBUILTIN_UDPv6_ALIAS
TRANSPORT_BUILTIN, 644
DDS_TransportBuiltinKind
TRANSPORT_BUILTIN, 643
DDS_TransportBuiltinKindMask
TRANSPORT_BUILTIN, 643
DDS_TransportBuiltinQosPolicy, 1408
mask, 1408
DDS_TransportInfo_t, 1409
class_id, 1409
message_size_max, 1409
DDS_TransportInfoSeq, 1410
DDS_TRANSPORTMULTICAST_QOS_POLICY_ID
All QoS Policies, 559
DDS_TRANSPORTMULTICAST_QOS_POLICY_NAME
TRANSPORT_MULTICAST, 646
DDS_TRANSPORTMULTICASTMAPPING_QOS_POLICY_ID
All QoS Policies, 559
DDS_TRANSPORTMULTICASTMAPPING_QOS_POLICY_NAME
TRANSPORT_MULTICAST_MAPPING, 647
DDS_TransportMulticastMapping_t, 1410
addresses, 1411
mapping_function, 1412
topic_expression, 1411
DDS_TransportMulticastMappingFunction_t, 1412
dll, 1412
function_name, 1413
DDS_TransportMulticastMappingQosPolicy, 1413
value, 1415
DDS_TransportMulticastMappingSeq, 1415
DDS_TransportMulticastQosPolicy, 1415
kind, 1416
value, 1416
DDS_TransportMulticastQosPolicyKind
TRANSPORT_MULTICAST, 646
DDS_TransportMulticastSettings_t, 1417
receive_address, 1418
receive_port, 1418
transports, 1418
DDS_TransportMulticastSettingsSeq, 1419
DDS_TRANSPORTPRIORITY_QOS_POLICY_ID
All QoS Policies, 558
DDS_TRANSPORTPRIORITY_QOS_POLICY_NAME
TRANSPORT_PRIORITY, 648
DDS_TransportPriorityQosPolicy, 1419
value, 1421
DDS_TRANSPORTSELECTION_QOS_POLICY_ID
All QoS Policies, 559
DDS_TRANSPORTSELECTION_QOS_POLICY_NAME
TRANSPORT_SELECTION, 648
DDS_TransportSelectionQosPolicy, 1421
enabled_transports, 1422
DDS_TRANSPORTUNICAST_QOS_POLICY_ID
All QoS Policies, 559
DDS_TRANSPORTUNICAST_QOS_POLICY_NAME
TRANSPORT_UNICAST, 649
DDS_TransportUnicastQosPolicy, 1422
value, 1424
DDS_TransportUnicastSettings_t, 1424
receive_port, 1425
transports, 1425
DDS_TransportUnicastSettingsSeq, 1426
DDS_TRUNCATE_PERSISTENT_JOURNAL
DURABILITY, 597
DDS_TrustAlgorithmBit
Built-in Topic's Trust Types, 101
DDS_TrustAlgorithmRequirements, 1426
Built-in Topic's Trust Types, 101
required_mask, 1426
supported_mask, 1426
DDS_TrustAlgorithmSet
Built-in Topic's Trust Types, 101
DDS_TYPE_CODE_PRINT_KIND_IDL
Type Code Support, 88
DDS_TYPE_CODE_PRINT_KIND_JSON_SCHEMA
Type Code Support, 88
DDS_TYPE_CODE_PRINT_KIND_XML
Type Code Support, 88
DDS_TYPE_CONSISTENCY_ENFORCEMENT_QOS_POLICY_ID
All QoS Policies, 558
DDS_TYPE_CONSISTENCY_ENFORCEMENT_QOS_POLICY_NAME
TYPE_CONSISTENCY_ENFORCEMENT, 651
DDS_TypeAllocationParams_t, 1427
allocate_optional_members, 1428
allocate_pointers, 1428
set_allocate_optional_members, 1428
set_allocate_pointers, 1427
DDS_TypeCode, 1428

add_member, 1463
add_member_ex, 1465
add_member_to_enum, 1461
add_member_to_union, 1462
array_dimension, 1450
array_dimension_count, 1449
cdr_serialized_sample_key_max_size, 1459, 1460
cdr_serialized_sample_max_size, 1458, 1459
cdr_serialized_sample_min_size, 1458, 1460
concrete_base_type, 1453
content_type, 1451
default_index, 1452
default_value, 1467
discriminator_type, 1447
element_count, 1450
equal, 1435
extensibility_kind, 1432
find_member_by_id, 1456
find_member_by_name, 1439
get_cdr_serialized_sample_max_size, 1457
get_type_object_serialized_size, 1457
is_alias_pointer, 1452
is_assignable_from, 1436
is_member_bitfield, 1445
is_member_key, 1443
is_member_must_understand, 1473
is_member_pointer, 1444
is_member_required, 1443
kind, 1432
length, 1448
max_value, 1468
member_bitfield_bits, 1445
member_count, 1437
member_default_value, 1470
member_id, 1455
member_label, 1441
member_label_count, 1440
member_max_value, 1471
member_min_value, 1470
member_name, 1438
member_ordinal, 1442
member_type, 1439
member_unit, 1472
member_visibility, 1446
min_value, 1467
name, 1436
print, 1474
print_IDL, 1474
to_string, 1475
type_modifier, 1454
unit, 1469
DDS_TYPECODE_INDEX_INVALID
Type Code Support, 83
DDS_TYPECODE_KEY_MEMBER
Type Code Support, 86
DDS_TYPECODE_MEMBER_ID_INVALID
Type Code Support, 83
DDS_TYPECODE_NONKEY_MEMBER
Type Code Support, 85
DDS_TYPECODE_NONKEY_REQUIRED_MEMBER
Type Code Support, 86
DDS_TYPECODE_NOT_BITFIELD
Type Code Support, 83
DDS_TypeCode_PrintFormat_INITIALIZER
Type Code Support, 87
DDS_TypeCodeFactory, 1476
clone_tc, 1479
create_alias_tc, 1486
create_array_tc, 1489
create_enum_tc, 1485, 1486
create_sequence_tc, 1488
create_string_tc, 1487
create_struct_tc, 1480, 1481
create_union_tc, 1483, 1484
create_value_tc, 1482, 1483
create_wstring_tc, 1487
delete_tc, 1479
get_instance, 1479
get_primitive_tc, 1480
DDS_TypeCodePrintFormatKind
Type Code Support, 88
DDS_TypeCodePrintFormatProperty, 1490
indent, 1490
print_complete_type, 1492
print_kind, 1492
print_ordinals, 1491
DDS_TypeConsistencyEnforcementQosPolicy, 1493
force_type_validation, 1496
ignore_enum_literal_names, 1496
ignore_member_names, 1495
ignore_sequence_bounds, 1495
ignore_string_bounds, 1495
kind, 1494
prevent_type_widening, 1496
DDS_TypeConsistencyKind
TYPE_CONSISTENCY_ENFORCEMENT, 650
DDS_TypeDeallocationParams_t, 1497
delete_optional_members, 1498
delete_pointers, 1498
set_delete_optional_members, 1498
set_delete_pointers, 1498
DDS_TYPESUPPORT_CPP
User Data Type Support, 74
DDS_TYPESUPPORT_QOS_POLICY_ID
All QoS Policies, 559
DDS_TYPESUPPORT_QOS_POLICY_NAME
TYPESUPPORT, 653
DDS_TypeSupportQosPolicy, 1499

cdr_padding_kind, 1500
plugin_data, 1500
DDS_UInt8
DDS-Specific Primitive Types, 514
DDS_UInt8Seq, 1500
DDS_UNICAST_ONLY_TRANSPORT_MULTICAST_QOS
TRANSPORT_MULTICAST, 646
DDS_UnionMember, 1501
is_pointer, 1502
labels, 1502
name, 1501
type, 1502
DDS_UnionMemberSeq, 1502
DDS_UNKNOWN_SAMPLE_IDENTITY
WriteParams, 676
DDS_UNKNOWN_SERVICE_REQUEST_ID
ServiceRequest Built-in Topic, 498
DDS_UNREGISTERED_INSTANCE_REPLACEMENT
DATA_WRITER_RESOURCE_LIMITS, 581
DDS_UnsignedLong
DDS-Specific Primitive Types, 515
DDS_UnsignedLongLong
DDS-Specific Primitive Types, 515
DDS_UnsignedLongLongSeq, 1503
DDS_UnsignedLongSeq, 1503
DDS_UnsignedShort
DDS-Specific Primitive Types, 514
DDS_UnsignedShortSeq, 1503
DDS_USER_EXCEPTION_CODE
Exception Codes, 530
DDS_USERDATA_QOS_POLICY_ID
All QoS Policies, 557
DDS_USERDATA_QOS_POLICY_NAME
USER_DATA, 653
DDS_UserDataQosPolicy, 1504
value, 1505
DDS_ValueMember, 1505
access, 1507
bits, 1507
id, 1507
is_key, 1507
is_must_understand, 1508
is_optional, 1507
is_pointer, 1507
name, 1506
type, 1506
DDS_ValueMemberSeq, 1508
DDS_ValueModifier
Type Code Support, 87
DDS_VENDOR_ID_LENGTH_MAX
Common types and functions, 502
DDS_VendorEndpointTrustAttributesMask
Built-in Topic's Trust Types, 101
DDS_VendorId_t, 1508
vendorId, 1509
DDS_ViewStateKind
View States, 163
DDS_ViewStateMask
View States, 163
DDS_VirtualWriterInfo, 1509
DDS_Visibility
Type Code Support, 88
DDS_VM_ABSTRACT
Type Code Support, 84
DDS_VM_CUSTOM
Type Code Support, 84
DDS_VM_NONE
Type Code Support, 84
DDS_VM_TRUNCATABLE
Type Code Support, 84
DDS_VOLATILE_DURABILITY_QOS
DURABILITY, 598
DDS_WaitSetProperty_t, 1509
max_event_count, 1510
max_event_delay, 1510
DDS_WaitSetProperty_t_INITIALIZER
Conditions and WaitSets, 670
DDS_WAL_PERSISTENT_JOURNAL
DURABILITY, 597
DDS_Wchar
DDS-Specific Primitive Types, 514
DDS_WcharSeq, 1510
DDS_WEB_INTEGRATION_SERVICE_QOS
SERVICE, 637
DDS_WIREPROTOCOL_QOS_POLICY_ID
All QoS Policies, 558
DDS_WIREPROTOCOL_QOS_POLICY_NAME
WIRE_PROTOCOL, 659
DDS_WireProtocolQosPolicy, 1511
check_crc, 1518
compute_crc, 1518
participant_id, 1514
rtps_app_id, 1516
rtps_auto_id_kind, 1518
rtps_host_id, 1515
rtps_instance_id, 1516
rtps_reserved_port_mask, 1517
rtps_well_known_ports, 1517
DDS_WireProtocolQosPolicyAutoKind
WIRE_PROTOCOL, 658
DDS_WRITEPARAMS_DEFAULT
WriteParams, 677
DDS_WriteParams_reset
WriteParams, 676
DDS_WriteParams_t, 1519
cookie, 1521
flag, 1522
handle, 1521

identity, 1520
priority, 1521
related_reader_guid, 1523
related_sample_identity, 1520
related_source_guid, 1523
replace_auto, 1520
source_guid, 1522
source_timestamp, 1521
DDS_WRITER_REMOVED_BATCH_SAMPLE
Sample Flags, 674
DDS_WRITERDATALIFECYCLE_QOS_POLICY_ID
All QoS Policies, 558
DDS_WRITERDATALIFECYCLE_QOS_POLICY_NAME
WRITER_DATA_LIFECYCLE, 654
DDS_WriterDataLifecycleQosPolicy, 1524
autodispose_unregistered_instances, 1525
autopurge_disposed_instances_delay, 1526
autopurge_unregistered_instances_delay, 1525
DDS_Wstring_alloc
String Support, 762
DDS_Wstring_copy
String Support, 763
DDS_Wstring_copy_and_widen
String Support, 764
DDS_Wstring_dup
String Support, 764
DDS_Wstring_dup_and_widen
String Support, 765
DDS_Wstring_free
String Support, 765
DDS_Wstring_length
String Support, 763
DDS_WstringSeq, 1527
DDS_XCDR2_DATA_REPRESENTATION
DATA_REPRESENTATION, 569
DDS_XCDR_DATA_REPRESENTATION
DATA_REPRESENTATION, 568
DDS_XML_DATA_REPRESENTATION
DATA_REPRESENTATION, 568
DDS_XML_PRINT_FORMAT
Topics, 67
DDS_ZERO_CDR_PADDING
TYPESUPPORT, 652
DDSAsyncWaitSet, 1528
~DDSAsyncWaitSet, 1533
attach_condition, 1536
attach_condition_with_completion_token, 1537
create_completion_token, 1540
DDSAsyncWaitSet, 1531, 1532
delete_completion_token, 1540
detach_condition, 1537
detach_condition_with_completion_token, 1538
get_conditions, 1540
get_property, 1539
start, 1533
start_with_completion_token, 1534
stop, 1535
stop_with_completion_token, 1535
unlock_condition, 1539
DDSAsyncWaitSetCompletionToken, 1541
wait, 1542
DDSAsyncWaitSetListener, 1543
on_thread_deleted, 1543
on_thread_spawned, 1543
on_wait_timeout, 1544
DDSCondition, 1544
dispatch, 1546
get_handler, 1546
get_trigger_value, 1545
set_handler, 1545
DDSConditionHandler, 1546
on_condition_triggered, 1547
DDSConditionSeq, 1547
DDSContentFilter, 1548
compile, 1548
evaluate, 1550
finalize, 1550
DDSContentFilteredTopic, 1551
append_to_expression_parameter, 1554
get_expression_parameters, 1553
get_filter_expression, 1552
get_related_topic, 1556
narrow, 1552
remove_from_expression_parameter, 1555
set_expression, 1554
set_expression_parameters, 1553
DDSDataReader, 1556
acknowledge_all, 1565, 1566
acknowledge_sample, 1564, 1565
create_querycondition, 1561
create_querycondition_w_params, 1562
create_readcondition, 1561
create_readcondition_w_params, 1561
create_topic_query, 1577
delete_contained_entities, 1563
delete_readcondition, 1562
delete_topic_query, 1578
enable, 1580
get_datareader_cache_status, 1573
get_datareader_protocol_status, 1573
get_instance_handle, 1582
get_listener, 1577
get_liveliness_changed_status, 1570
get_matched_publication_data, 1568
get_matched_publication_datareader_protocol_status, 1573
get_matched_publication_participant_data, 1569
get_matched_publications, 1567

get_qos, 1575
get_requested_deadline_missed_status, 1571
get_requested_incompatible_qos_status, 1571
get_sample_lost_status, 1572
get_sample_rejected_status, 1570
get_status_changes, 1582
get_statuscondition, 1581
get_subscriber, 1570
get_subscription_matched_status, 1572
get_topicdescription, 1569
is_matched_publication_alive, 1568
lookup_topic_query, 1578
set_listener, 1577
set_property, 1576
set_qos, 1574
set_qos_with_profile, 1575
take_discovery_snapshot, 1579
wait_for_historical_data, 1564
DDSDataReaderListener, 1583
on_data_available, 1585
on_liveliness_changed, 1584
on_requested_deadline_missed, 1584
on_requested_incompatible_qos, 1584
on_sample_lost, 1585
on_sample_rejected, 1584
on_subscription_matched, 1585
DDSDataReaderSeq, 1585
DDSDataReaderStatusConditionHandler, 1586
~DDSDataReaderStatusConditionHandler, 1587
DDSDataReaderStatusConditionHandler, 1587
on_condition_triggered, 1588
DDSDataTagQosPolicyHelper, 1588
DDSDataWriter, 1589
assert_liveliness, 1597
enable, 1610
flush, 1607
get_datawriter_cache_status, 1594
get_datawriter_protocol_status, 1595
get_instance_handle, 1611
get_listener, 1607
get_liveliness_lost_status, 1592
get_matched_subscription_data, 1600
get_matched_subscription_datawriter_protocol_status, 1595
get_matched_subscription_datawriter_protocol_status_by_locator, 1597
get_matched_subscription_locators, 1598
get_matched_subscription_participant_data, 1601
get_matched_subscriptions, 1599
get_offered_deadline_missed_status, 1592
get_offered_incompatible_qos_status, 1593
get_publication_matched_status, 1593
get_publisher, 1602
get_qos, 1606
get_reliable_reader_activity_changed_status, 1594
get_reliable_writer_cache_changed_status, 1594
get_service_request_accepted_status, 1596
get_status_changes, 1611
get_statuscondition, 1611
get_topic, 1601
is_matched_subscription_active, 1596
is_sample_app_acknowledged, 1603
set_listener, 1607
set_property, 1605
set_qos, 1604
set_qos_with_profile, 1605
take_discovery_snapshot, 1608, 1609
wait_for_acknowledgments, 1602
wait_for_asynchronous_publishing, 1603
DDSDataWriterListener, 1612
on_application_acknowledgment, 1617
on_instance_replaced, 1617
on_liveliness_lost, 1614
on_offered_deadline_missed, 1613
on_offered_incompatible_qos, 1614
on_publication_matched, 1615
on_reliable_reader_activity_changed, 1616
on_reliable_writer_cache_changed, 1615
on_sample_removed, 1616
on_service_request_accepted, 1618
DDSDomainEntity, 1618
DDSDomainParticipant, 1619
add_peer, 1678
assert_liveliness, 1667
banish_ignored_participants, 1662
contains_entity, 1674
create_contentfilteredtopic, 1653
create_contentfilteredtopic_with_filter, 1654
create_datareader, 1687
create_datareader_with_profile, 1688
create_datawriter, 1683
create_datawriter_with_profile, 1685
create_flowcontroller, 1659
create_multitopic, 1656
create_publisher, 1644
create_publisher_with_profile, 1645
create_subscriber, 1647
create_subscriber_with_profile, 1648
create_topic, 1650
create_topic_with_profile, 1651
delete_contained_entities, 1669
delete_contentfilteredtopic, 1655
delete_datareader, 1689
delete_datawriter, 1686
delete_durable_subscription, 1667
delete_flowcontroller, 1660
delete_multitopic, 1657
delete_publisher, 1646

delete_subscriber, 1648
delete_topic, 1653
find_topic, 1657
get_builtin_subscriber, 1661
get_current_time, 1666
get_default_datareader_qos, 1628
get_default_datawriter_qos, 1626
get_default_flowcontroller_property, 1630
get_default_library, 1634
get_default_profile, 1634
get_default_profile_library, 1634
get_default_publisher_qos, 1639
get_default_subscriber_qos, 1641
get_default_topic_qos, 1636
get_discovered_participant_data, 1671
get_discovered_participant_subject_name, 1672
get_discovered_participants, 1670
get_discovered_participants_from_subject_name, 1670
get_discovered_topic_data, 1673
get_discovered_topics, 1673
get_dns_tracker_polling_period, 1680
get_domain_id, 1665
get_implicit_publisher, 1682
get_implicit_subscriber, 1683
get_listener, 1682
get_participant_protocol_status, 1676
get_publishers, 1649
get_qos, 1678
get_subscribers, 1650
get_typecode, 1675
ignore_participant, 1661
ignore_publication, 1664
ignore_subscription, 1664
ignore_topic, 1663
lookup_contentfilter, 1633
lookup_datareader_by_name, 1692
lookup_datawriter_by_name, 1691
lookup_flowcontroller, 1660
lookup_publisher_by_name, 1690
lookup_subscriber_by_name, 1690
lookup_topicdescription, 1658
register_contentfilter, 1632
register_durable_subscription, 1666
remove_peer, 1679
resume_endpoint_discovery, 1668
set_default_datareader_qos, 1629
set_default_datareader_qos_with_profile, 1630
set_default_datawriter_qos, 1627
set_default_datawriter_qos_with_profile, 1627
set_default_flowcontroller_property, 1631
set_default_library, 1635
set_default_profile, 1636
set_default_publisher_qos, 1640
set_default_publisher_qos_with_profile, 1640
set_default_subscriber_qos, 1642
set_default_subscriber_qos_with_profile, 1643
set_default_topic_qos, 1637
set_default_topic_qos_with_profile, 1638
set_dns_tracker_polling_period, 1681
set_listener, 1682
set_property, 1676
set_qos, 1677
set_qos_with_profile, 1677
take_discovery_snapshot, 1693
unregister_contentfilter, 1633
DDSDomainParticipantFactory, 1694
create_participant, 1710
create_participant_from_config, 1717
create_participant_from_config_w_params, 1718
create_participant_with_profile, 1711
delete_participant, 1713
finalize_instance, 1697
get_datareader_qos_from_profile, 1707
get_datareader_qos_from_profile_w_topic_name, 1708
get_datawriter_qos_from_profile, 1706
get_datawriter_qos_from_profile_w_topic_name, 1706
get_default_library, 1701
get_default_participant_qos, 1700
get_default_profile, 1703
get_default_profile_library, 1703
get_instance, 1697
get_participant_factory_qos_from_profile, 1703
get_participant_qos_from_profile, 1704
get_participants, 1721
get_publisher_qos_from_profile, 1705
get_qos, 1715
get_qos_profile_libraries, 1710
get_qos_profiles, 1710
get_subscriber_qos_from_profile, 1705
get_topic_qos_from_profile, 1708
get_topic_qos_from_profile_w_topic_name, 1709
get_typecode_from_config, 1717
load_profiles, 1715
lookup_participant, 1713
lookup_participant_by_name, 1719
register_type_support, 1720
reload_profiles, 1715
set_default_library, 1701
set_default_participant_qos, 1698
set_default_participant_qos_with_profile, 1699
set_default_profile, 1702
set_qos, 1714
set_thread_factory, 1721
unload_profiles, 1716
unregister_thread, 1716

DDSDomainParticipantFactory_RegisterTypeFunction
DomainParticipantFactory, 44
DDSDomainParticipantListener, 1722
on_invalid_local_status_advance_notice, 1724
DDSDynamicDataReader, 1724
DDSDynamicDataTypeSupport, 1725
~DDSDynamicDataTypeSupport, 1726
copy_data, 1729
create_data, 1728
DDSDynamicDataTypeSupport, 1726
delete_data, 1729
get_data_type, 1728
get_type_name, 1728
is_valid, 1727
print_data, 1729
register_type, 1727
unregister_type, 1727
DDSDynamicDataWriter, 1730
DDSEntity, 1731
enable, 1734
get_instance_handle, 1736
get_status_changes, 1735
get_statuscondition, 1735
DDSFlowController, 1736
get_name, 1739
get_participant, 1739
get_property, 1738
set_property, 1737
trigger_flow, 1738
DDSGuardCondition, 1739
~DDSGuardCondition, 1740
DDSGuardCondition, 1740
get_trigger_value, 1741
set_trigger_value, 1741
DDSKeyedOctetsDataReader, 1741
get_key_value, 1748, 1749
lookup_instance, 1749
narrow, 1750
read, 1744
read_instance, 1745
read_instance_w_condition, 1746
read_next_instance, 1747
read_next_instance_w_condition, 1747
read_next_sample, 1745
read_w_condition, 1744
return_loan, 1748
take, 1744
take_instance, 1746
take_instance_w_condition, 1746
take_next_instance, 1747
take_next_instance_w_condition, 1748
take_next_sample, 1745
take_w_condition, 1744
DDSKeyedOctetsDataWriter, 1750
create_data, 1756
delete_data, 1756
dispose, 1761
dispose_w_timestamp, 1761
get_key_value, 1762
lookup_instance, 1762, 1763
narrow, 1753
register_instance, 1753
register_instance_w_timestamp, 1754
unregister_instance, 1754, 1755
unregister_instance_w_timestamp, 1755
write, 1756, 1757
write_w_params, 1759, 1760
write_w_timestamp, 1758, 1759
DDSKeyedOctetsTypeSupport, 1763
data_to_string, 1769
deserialize_data_from_cdr_buffer, 1768
get_type_name, 1767
get_typecode, 1768
print_data, 1767
register_type, 1764
serialize_data_to_cdr_buffer, 1768
serialize_data_to_cdr_buffer_ex, 1768
unregister_type, 1766
DDSKeyedStringDataReader, 1769
get_key_value, 1776, 1777
lookup_instance, 1777
narrow, 1778
read, 1772
read_instance, 1773
read_instance_w_condition, 1774
read_next_instance, 1775
read_next_instance_w_condition, 1775
read_next_sample, 1773
read_w_condition, 1772
return_loan, 1776
take, 1772
take_instance, 1774
take_instance_w_condition, 1774
take_next_instance, 1775
take_next_instance_w_condition, 1776
take_next_sample, 1773
take_w_condition, 1772
DDSKeyedStringDataWriter, 1778
create_data, 1783
delete_data, 1783
dispose, 1786
dispose_w_timestamp, 1786, 1787
get_key_value, 1787
lookup_instance, 1788
narrow, 1780
register_instance, 1781
register_instance_w_timestamp, 1781
unregister_instance, 1782

unregister_instance_w_timestamp, 1782, 1783
write, 1784
write_w_params, 1785
write_w_timestamp, 1784, 1785
DDSKeyedStringTypeSupport, 1788
data_to_string, 1794
deserialize_data_from_cdr_buffer, 1793
get_type_name, 1792
get_typecode, 1793
print_data, 1792
register_type, 1789
serialize_data_to_cdr_buffer, 1793
serialize_data_to_cdr_buffer_ex, 1793
unregister_type, 1791
DDSListener, 1794
DDSMultiTopic, 1797
get_expression_parameters, 1799
get_subscription_expression, 1799
narrow, 1799
set_expression_parameters, 1800
DDSOctetsDataReader, 1800
narrow, 1804
read, 1802
read_next_sample, 1803
read_w_condition, 1802
return_loan, 1803
take, 1802
take_next_sample, 1803
take_w_condition, 1802
DDSOctetsDataWriter, 1804
create_data, 1806
delete_data, 1806
narrow, 1806
write, 1807, 1808
write_w_params, 1809, 1810
write_w_timestamp, 1808, 1809
DDSOctetsTypeSupport, 1811
data_to_string, 1815
deserialize_data_from_cdr_buffer, 1815
get_type_name, 1813
get_typecode, 1814
print_data, 1814
register_type, 1812
serialize_data_to_cdr_buffer, 1814
serialize_data_to_cdr_buffer_ex, 1814
unregister_type, 1812
DDSParticipantBuiltinTopicDataDataReader, 1816
DDSParticipantBuiltinTopicDataTypeSupport, 1816
DDSPropertyQosPolicyHelper, 1817
DDSPublicationBuiltinTopicDataDataReader, 1817
DDSPublicationBuiltinTopicDataTypeSupport, 1818
DDSPublisher, 1818
begin_coherent_changes, 1832
copy_from_topic_qos, 1834
create_datawriter, 1826
create_datawriter_with_profile, 1827
delete_contained_entities, 1833
delete_datawriter, 1829
end_coherent_changes, 1833
get_all_datawriters, 1830
get_default_datawriter_qos, 1821
get_default_library, 1824
get_default_profile, 1825
get_default_profile_library, 1826
get_listener, 1838
get_participant, 1833
get_qos, 1837
lookup_datawriter, 1829
lookup_datawriter_by_name, 1839
resume_publications, 1831
set_default_datawriter_qos, 1822
set_default_datawriter_qos_with_profile, 1823
set_default_library, 1823
set_default_profile, 1824
set_listener, 1838
set_qos, 1836
set_qos_with_profile, 1837
suspend_publications, 1830
wait_for_acknowledgments, 1835
wait_for_asynchronous_publishing, 1835
DDSPublisherListener, 1839
DDSPublisherSeq, 1840
DDSQueryCondition, 1841
get_query_expression, 1842
get_query_parameters, 1842
set_query_parameters, 1842
DDSReadCondition, 1842
get_datareader, 1844
get_instance_state_mask, 1844
get_sample_state_mask, 1843
get_stream_kind_mask, 1844
get_view_state_mask, 1844
DDSServiceRequestDataReader, 1845
DDSServiceRequestTypeSupport, 1845
DDSStatusCondition, 1846
get_enabled_statuses, 1846
get_entity, 1847
set_enabled_statuses, 1847
DDSStringDataReader, 1848
narrow, 1851
read, 1849
read_next_sample, 1850
read_w_condition, 1849
return_loan, 1851
take, 1849
take_next_sample, 1850
take_w_condition, 1850
DDSStringDataWriter, 1852

create_data, 1853
delete_data, 1853
narrow, 1853
write, 1854
write_w_params, 1854
write_w_timestamp, 1854
DDSStringTypeSupport, 1855
data_to_string, 1859
deserialize_data_from_cdr_buffer, 1859
get_type_name, 1857
get_typecode, 1858
print_data, 1858
register_type, 1856
serialize_data_to_cdr_buffer, 1858
serialize_data_to_cdr_buffer_ex, 1859
unregister_type, 1857
DDSSubscriber, 1860
begin_access, 1873
copy_from_topic_qos, 1876
create_datareader, 1867
create_datareader_with_profile, 1869
delete_contained_entities, 1871
delete_datareader, 1870
end_access, 1874
get_all_datareaders, 1875
get_datareaders, 1874
get_default_datareader_qos, 1863
get_default_library, 1866
get_default_profile, 1867
get_default_profile_library, 1867
get_listener, 1880
get_participant, 1876
get_qos, 1879
lookup_datareader, 1872
lookup_datareader_by_name, 1880
notify_datareaders, 1876
set_default_datareader_qos, 1863
set_default_datareader_qos_with_profile, 1864
set_default_library, 1865
set_default_profile, 1866
set_listener, 1879
set_qos, 1877
set_qos_with_profile, 1878
DDSSubscriberListener, 1881
on_data_on_readers, 1882
DDSSubscriberSeq, 1882
DDSSubscriptionBuiltinTopicDataDataReader, 1882
DDSSubscriptionBuiltinTopicDataTypeSupport, 1883
DDSTheParticipantFactory
DomainParticipantFactory, 43
DDSThreadFactory, 1883
create_thread, 1884
delete_thread, 1884
DDSThreadFactory_OnSpawnedFunction
User-managed Threads, 718
DDSTopic, 1885
get_inconsistent_topic_status, 1887
get_listener, 1890
get_qos, 1889
narrow, 1887
set_listener, 1889
set_qos, 1887
set_qos_with_profile, 1888
DDSTopicBuiltinTopicDataDataReader, 1890
DDSTopicBuiltinTopicDataTypeSupport, 1891
DDSTopicDescription, 1892
get_name, 1893
get_participant, 1893
get_type_name, 1892
DDSTopicListener, 1894
on_inconsistent_topic, 1895
DDSTopicQuery, 1895
get_guid, 1895
DDSTopicQueryHelper, 1896
topic_query_data_from_service_request, 1896
DDSTypeSupport, 1897
DDSWaitSet, 1897
∼DDSWaitSet, 1900
attach_condition, 1902
DDSWaitSet, 1901
detach_condition, 1903
get_conditions, 1903
get_property, 1904
set_property, 1904
wait, 1901
DDSWriterContentFilter, 1905
writer_attach, 1908
writer_compile, 1906
writer_detach, 1909
writer_evaluate, 1907
writer_finalize, 1908
writer_return_loan, 1909
DEADLINE, 583
DDS_DEADLINE_QOS_POLICY_NAME, 584
deadline
DDS_DataReaderQos, 861
DDS_DataWriterQos, 912
DDS_PublicationBuiltinTopicData, 1267
DDS_SubscriptionBuiltinTopicData, 1373
DDS_TopicBuiltinTopicData, 1393
DDS_TopicQos, 1400
debug
NDDSConfigLogger, 2100
Debuggability, 266
dedicated_participant
DDS_MonitoringDistributionSettings, 1196
default_domain_announcement_period
DDS_DiscoveryConfigQosPolicy, 949

default_index
DDS_TypeCode, 1452
default_unicast
DDS_DomainParticipantQos, 974
default_unicast_locators
DDS_ParticipantBuiltinTopicData, 1232
default_value
DDS_TypeCode, 1467
delete_completion_token
DDSAsyncWaitSet, 1540
delete_contained_entities
DDSDataReader, 1563
DDSDomainParticipant, 1669
DDSPublisher, 1833
DDSSubscriber, 1871
delete_contentfilteredtopic
DDSDomainParticipant, 1655
delete_data
DDSDynamicDataTypeSupport, 1729
DDSKeyedOctetsDataWriter, 1756
DDSKeyedStringDataWriter, 1783
DDSOctetsDataWriter, 1806
DDSStringDataWriter, 1853
FooDataWriter, 1962
FooTypeSupport, 1984
rti::flat::Sample< OffsetType >, 2190
delete_data_ex
FooTypeSupport, 1985
delete_datareader
DDSDomainParticipant, 1689
DDSSubscriber, 1870
delete_datawriter
DDSDomainParticipant, 1686
DDSPublisher, 1829
delete_durable_subscription
DDSDomainParticipant, 1667
delete_flowcontroller
DDSDomainParticipant, 1660
delete_multitopic
DDSDomainParticipant, 1657
delete_optional_members
DDS_TypeDeallocationParams_t, 1498
delete_participant
DDSDomainParticipantFactory, 1713
delete_pointers
DDS_TypeDeallocationParams_t, 1498
delete_publisher
DDSDomainParticipant, 1646
delete_readcondition
DDSDataReader, 1562
delete_subscriber
DDSDomainParticipant, 1648
delete_tc
DDS_TypeCodeFactory, 1479
delete_thread
DDSThreadFactory, 1884
delete_topic
DDSDomainParticipant, 1653
delete_topic_query
DDSDataReader, 1578
deny_interfaces_list
NDDS_Transport_Property_t, 2048
deny_interfaces_list_length
NDDS_Transport_Property_t, 2049
deny_multicast_interfaces_list
NDDS_Transport_Property_t, 2050
deny_multicast_interfaces_list_length
NDDS_Transport_Property_t, 2050
Deprecated QoS, 384
depth
DDS_HistoryQosPolicy, 1164
deserialize_data_from_cdr_buffer
DDSKeyedOctetsTypeSupport, 1768
DDSKeyedStringTypeSupport, 1793
DDSOctetsTypeSupport, 1815
DDSStringTypeSupport, 1859
FooTypeSupport, 1990
deserialized_type_object_dynamic_allocation_threshold
DDS_DomainParticipantResourceLimitsQosPolicy, 996
DESTINATION_ORDER, 584
DDS_BY_RECEPTION_TIMESTAMP_DESTINATIONORDER_QOS, 585
DDS_BY_SOURCE_TIMESTAMP_DESTINATIONORDER_QOS, 585
DDS_DESTINATIONORDER_QOS_POLICY_NAME, 586
DDS_DestinationOrderQosPolicyKind, 585
DDS_DestinationOrderQosPolicyScopeKind, 586
DDS_INSTANCE_SCOPE_DESTINATIONORDER_QOS, 586
DDS_TOPIC_SCOPE_DESTINATIONORDER_QOS, 586
destination_order
DDS_DataReaderQos, 862
DDS_DataWriterQos, 913
DDS_PublicationBuiltinTopicData, 1268
DDS_SubscriptionBuiltinTopicData, 1374
DDS_TopicBuiltinTopicData, 1394
DDS_TopicQos, 1401
detach_condition
DDSAsyncWaitSet, 1537
DDSWaitSet, 1903
detach_condition_with_completion_token
DDSAsyncWaitSet, 1538
detached_instance_count
DDS_DataReaderCacheStatus, 842
detached_instance_count_peak

DDS_DataReaderCacheStatus, 842
difference_type
rti::flat::SequenceIterator< E, OffsetKind >, 2201
direct_communication
DDS_DurabilityQosPolicy, 1007
disable
NDDSUtilityHeapMonitoring, 2114
NDDSUtilityNetworkCapture, 2119
disable_asynchronous_batch
DDS_AsynchronousPublisherQosPolicy, 800
disable_asynchronous_write
DDS_AsynchronousPublisherQosPolicy, 800
disable_fragmentation_support
DDS_BuiltinTopicReaderResourceLimits_t, 818
DDS_DataReaderResourceLimitsQosPolicy, 874
disable_heap_monitoring
NDDSUtility, 2112
disable_inline_keyhash
DDS_DataWriterProtocolQosPolicy, 895
disable_interface_tracking
NDDS_Transport_UDPv4_Property_t, 2065
NDDS_Transport_UDPv4_WAN_Property_t, 2073
NDDS_Transport_UDPv6_Property_t, 2085
disable_positive_acks
DDS_DataReaderProtocolQosPolicy, 845
DDS_DataWriterProtocolQosPolicy, 894
DDS_PublicationBuiltinTopicData, 1272
DDS_SubscriptionBuiltinTopicData, 1378
disable_positive_acks_decrease_sample_keep_duration_factor
DDS_RtpsReliableWriterProtocol_t, 1332
disable_positive_acks_enable_adaptive_sample_keep_duration
DDS_RtpsReliableWriterProtocol_t, 1331
disable_positive_acks_increase_sample_keep_duration_factor
DDS_RtpsReliableWriterProtocol_t, 1332
disable_positive_acks_max_sample_keep_duration
DDS_RtpsReliableWriterProtocol_t, 1331
disable_positive_acks_min_sample_keep_duration
DDS_RtpsReliableWriterProtocol_t, 1330
disable_repair_piggyback_heartbeat
DDS_RtpsReliableWriterProtocol_t, 1338
disable_topic_query_publication
DDS_AsynchronousPublisherQosPolicy, 801
disabled_metrics_selection
DDS_MonitoringMetricSelection, 1207
discard
rti::flat::AbstractBuilder, 788
discard_builder
FlatData Builders, 768
discard_loan
FooDataWriter, 1964
DISCOVERY, 586
DDS_DISCOVERY_QOS_POLICY_NAME, 587
Discovery, 271
discovery
DDS_DomainParticipantQos, 974
Discovery Isolation, 277
Discovery Metadata, 377
DISCOVERY_CONFIG, 587
DDS_DISCOVERYCONFIG_BUILTIN_CHANNEL_MASK_ALL, 590
DDS_DISCOVERYCONFIG_BUILTIN_CHANNEL_MASK_DEFAULT, 589
DDS_DISCOVERYCONFIG_BUILTIN_CHANNEL_MASK_NONE, 589
DDS_DISCOVERYCONFIG_BUILTIN_DPSE, 592
DDS_DISCOVERYCONFIG_BUILTIN_PLUGIN_MASK_DEFAULT, 589
DDS_DISCOVERYCONFIG_BUILTIN_PLUGIN_MASK_NONE, 589
DDS_DISCOVERYCONFIG_BUILTIN_SDP, 591
DDS_DISCOVERYCONFIG_BUILTIN_SDP2, 592
DDS_DISCOVERYCONFIG_BUILTIN_SEDP, 591
DDS_DISCOVERYCONFIG_BUILTIN_SPDP, 591
DDS_DISCOVERYCONFIG_BUILTIN_SPDP2, 592
DDS_DISCOVERYCONFIG_QOS_POLICY_NAME, 594
DDS_DISCOVERYCONFIG_SERVICE_REQUEST_CHANNEL, 592
DDS_DISCOVERYCONFIG_TYPE_LOOKUP_SERVICE_CHANNEL, 592
DDS_DiscoveryConfigBuiltinChannelKind, 592
DDS_DiscoveryConfigBuiltinChannelKindMask, 590
DDS_DiscoveryConfigBuiltinPluginKind, 591
DDS_DiscoveryConfigBuiltinPluginKindMask, 590
DDS_LIVELINESS_BASED_REMOTE_PARTICIPANT_PURGE, 593
DDS_NO_REMOTE_PARTICIPANT_PURGE, 594
DDS_RemoteParticipantPurgeKind, 593
discovery_config
DDS_DomainParticipantQos, 975
discriminator_type
DDS_TypeCode, 1447
dispatch
DDSCondition, 1546
dispose
DDSKeyedOctetsDataWriter, 1761
DDSKeyedStringDataWriter, 1786
FooDataWriter, 1957
dispose_w_params
FooDataWriter, 1959
dispose_w_timestamp
DDSKeyedOctetsDataWriter, 1761
DDSKeyedStringDataWriter, 1786, 1787
FooDataWriter, 1958
disposed_generation_count
DDS_SampleInfo, 1349
disposed_instance_count
DDS_DataReaderCacheStatus, 841

DDS_DataWriterCacheStatus, 891
disposed_instance_count_peak
DDS_DataReaderCacheStatus, 841
DDS_DataWriterCacheStatus, 891
disposed_instance_removal
DDS_DataReaderResourceLimitsInstanceReplacementSettings, 867
distribution_settings
DDS_MonitoringQosPolicy, 1212
dll
DDS_TransportMulticastMappingFunction_t, 1412
dns_tracker_polling_period
DDS_DiscoveryConfigQosPolicy, 956
Documentation Roadmap, 430
Domain Module, 41
domain_entity_qos_library_name
DDS_DomainParticipantConfigParams_t, 966
domain_entity_qos_profile_name
DDS_DomainParticipantConfigParams_t, 966
domain_id
DDS_DomainParticipantConfigParams_t, 965
DDS_MonitoringDedicatedParticipantSettings, 1194
DDS_ParticipantBuiltinTopicData, 1232
domain_id_gain
DDS_RtpsWellKnownPorts_t, 1341
DOMAIN_PARTICIPANT_RESOURCE_LIMITS, 594
DDS_AUTO_COUNT, 595
DDS_DOMAINPARTICIPANTRESOURCELIMITS_QOS_POLICY_NAME, 596
DDS_DomainParticipantResourceLimitsIgnoredEntityReplacementKind, 595
DDS_NO_REPLACEMENT_IGNORED_ENTITY_REPLACEMENT, 595
DDS_NOT_ALIVE_FIRST_IGNORED_ENTITY_REPLACEMENT, 595
DomainParticipantConfigParams, 716
DDS_DOMAIN_ID_USE_XML_CONFIG, 717
DDS_DomainParticipantConfigParams_t_INITIALIZER, 717
DDS_ENTITY_NAME_USE_XML_CONFIG, 717
DDS_QOS_ELEMENT_NAME_USE_XML_CONFIG, 717
DomainParticipantFactory, 42
DDS_DomainParticipantFactoryQos_equals, 44
DDS_PARTICIPANT_CONFIG_PARAMS_DEFAULT, 50
DDS_PARTICIPANT_QOS_DEFAULT, 50
DDSDomainParticipantFactory_RegisterTypeFunction, 44
DDSTheParticipantFactory, 43
print, 45
to_string, 45–49
DomainParticipants, 51
DDS_DomainId_t, 53
DDS_DomainParticipantQos_equals, 53
DDS_FLOW_CONTROLLER_PROPERTY_DEFAULT, 60
DDS_PUBLISHER_QOS_DEFAULT, 59
DDS_SQLFILTER_NAME, 61
DDS_STRINGMATCHFILTER_NAME, 61
DDS_SUBSCRIBER_QOS_DEFAULT, 59
DDS_TOPIC_QOS_DEFAULT, 58
print, 53
to_string, 54–57
drop_incomplete_coherent_set
DDS_PresentationQosPolicy, 1253
dropped_content
NDDS_Utility_NetworkCaptureParams_t, 2088
dropped_fragment_count
DDS_DataReaderProtocolStatus, 857
duplicate_sample_bytes
DDS_DataReaderProtocolStatus, 851
duplicate_sample_bytes_change
DDS_DataReaderProtocolStatus, 851
duplicate_sample_count
DDS_DataReaderProtocolStatus, 851
duplicate_sample_count_change
DDS_DataReaderProtocolStatus, 851
DURABILITY, 596
DDS_AUTO_WRITER_DEPTH, 599
DDS_DELETE_PERSISTENT_JOURNAL, 597
DDS_DURABILITY_QOS_POLICY_NAME, 599
DDS_DurabilityQosPolicyKind, 598
DDS_KUL_PERSISTENT_SYNCHRONIZATION, 598
DDS_MEMORY_PERSISTENT_JOURNAL, 597
DDS_NORMAL_PERSISTENT_SYNCHRONIZATION, 598
DDS_OFF_PERSISTENT_JOURNAL, 597
DDS_OFF_PERSISTENT_SYNCHRONIZATION, 598
DDS_PERSIST_PERSISTENT_JOURNAL, 597
DDS_PERSISTENT_DURABILITY_QOS, 599
DDS_PersistentJournalKind, 597
DDS_PersistentSynchronizationKind, 597
DDS_TRANSIENT_DURABILITY_QOS, 599
DDS_TRANSIENT_LOCAL_DURABILITY_QOS, 599
DDS_TRUNCATE_PERSISTENT_JOURNAL, 597
DDS_VOLATILE_DURABILITY_QOS, 598
DDS_WAL_PERSISTENT_JOURNAL, 597
Durability, 278
durability
DDS_DataReaderQos, 861
DDS_DataWriterQos, 912
DDS_PublicationBuiltinTopicData, 1266
DDS_SubscriptionBuiltinTopicData, 1373
DDS_TopicBuiltinTopicData, 1393

DDS_TopicQos, 1400
Durability and Persistence, 196
DURABILITY_SERVICE, 600
DDS_DURABILITYSERVICE_QOS_POLICY_NAME, 600
durability_service
DDS_DataWriterQos, 912
DDS_PublicationBuiltinTopicData, 1267
DDS_TopicBuiltinTopicData, 1393
DDS_TopicQos, 1400
duration
DDS_LatencyBudgetQosPolicy, 1175
DDS_LifespanQosPolicy, 1176
Dynamic Data, 103
DDS_DYNAMIC_DATA_JSON_PARSER_PROPERTIES_DEFAULT, 106
DDS_DYNAMIC_DATA_MEMBER_ID_UNSPECIFIED, 105
DDS_DYNAMIC_DATA_PROPERTY_DEFAULT, 105
DDS_DYNAMIC_DATA_TYPE_PROPERTY_DEFAULT, 106
DDS_DynamicDataMemberId, 105
dynamically_allocate_fragmented_samples
DDS_BuiltinTopicReaderResourceLimits_t, 821
DDS_DataReaderResourceLimitsQosPolicy, 876

element_count
DDS_DynamicDataMemberInfo, 1136
DDS_TypeCode, 1450
rti::flat::AbstractListBuilder< DHeaderGen >, 790
rti::flat::AbstractSequenceBuilder< DHeaderGen >, 792
rti::flat::PrimitiveArrayOffset< T, N >, 2133
rti::flat::PrimitiveSequenceOffset< T >, 2139
rti::flat::SequenceOffset< ElementOffset >, 2206
rti::flat::StringOffset, 2217
element_kind
DDS_DynamicDataMemberInfo, 1136
emergency
NDDSConfigLogger, 2098
empty_reliable_writer_cache
DDS_ReliableWriterCacheChangedStatus, 1303
enable
DDS_BatchQosPolicy, 810
DDS_MonitoringDedicatedParticipantSettings, 1193
DDS_MonitoringQosPolicy, 1211
DDS_PersistentStorageSettings, 1244
DDS_TopicQueryDispatchQosPolicy, 1405
DDSDataReader, 1580
DDSDataWriter, 1610
DDSEntity, 1734
NDDSUtilityHeapMonitoring, 2114
NDDSUtilityNetworkCapture, 2118
enable_data_consistency_check
DDS_DataWriterShmemRefTransferModeSettings, 927
enable_endpoint_discovery
DDS_DiscoveryQosPolicy, 964
enable_heap_monitoring
NDDSUtility, 2112
enable_multicast_periodic_heartbeat
DDS_RtpsReliableWriterProtocol_t, 1337
enable_required_subscriptions
DDS_AvailabilityQosPolicy, 808
enable_udp_debugging
NDDS_Transport_Shmem_Property_t, 2054
enable_v4mapped
NDDS_Transport_UDPv6_Property_t, 2082
enabled_builtin_channels
DDS_DiscoveryConfigQosPolicy, 946
enabled_metrics_selection
DDS_MonitoringMetricSelection, 1207
enabled_transports
DDS_DiscoveryQosPolicy, 961
DDS_TransportSelectionQosPolicy, 1422
end
connext::LoanedSamples< T >, 2003, 2004
rti::flat::AbstractAlignedList< ElementOffset >, 786
end_access
DDSSubscriber, 1874
end_coherent_changes
DDSPublisher, 1833
endpoint_type_object_lb_serialization_threshold
DDS_DiscoveryConfigQosPolicy, 956
ensure_length
FooSeq, 1974
Entity Support, 669
Entity Use Cases, 406
ENTITY_FACTORY, 600
DDS_ENTITYFACTORY_QOS_POLICY_NAME, 601
entity_factory
DDS_DomainParticipantFactoryQos, 969
DDS_DomainParticipantQos, 974
DDS_PublisherQos, 1278
DDS_SubscriberQos, 1369
ENTITY_NAME, 601
DDS_ENTITYNAME_QOS_POLICY_NAME, 602
enum_as_int
DDS_PrintFormatProperty, 1254
equal
DDS_DynamicData, 1032
DDS_TypeCode, 1435
error
NDDSConfigLogger, 2099
evaluate
DDSContentFilter, 1550
EVENT, 602
DDS_EVENT_QOS_POLICY_NAME, 603

event
DDS_DomainParticipantQos, 975
event_settings
DDS_MonitoringDistributionSettings, 1196
exact_reliability_kind_match
DDS_ReliabilityQosPolicy, 1299
Exception Codes, 529
DDS_BAD_MEMBER_ID_USER_EXCEPTION_CODE, 531
DDS_BAD_MEMBER_NAME_USER_EXCEPTION_CODE, 531
DDS_BAD_PARAM_SYSTEM_EXCEPTION_CODE, 530
DDS_BAD_TYPECODE_SYSTEM_EXCEPTION_CODE, 530
DDS_BADKIND_USER_EXCEPTION_CODE, 530
DDS_BOUNDS_USER_EXCEPTION_CODE, 530
DDS_ExceptionCode_t, 530
DDS_IMMUTABLE_TYPECODE_SYSTEM_EXCEPTION_CODE, 530
DDS_NO_EXCEPTION_CODE, 530
DDS_NO_MEMORY_SYSTEM_EXCEPTION_CODE, 530
DDS_SYSTEM_EXCEPTION_CODE, 530
DDS_USER_EXCEPTION_CODE, 530
ExceptionHelper.hpp, 2298
expects_inline_qos
DDS_DataReaderProtocolQosPolicy, 844
expiration_code
DDS_InvalidLocalAdvanceNoticeStatus, 1168
expiration_message
DDS_InvalidLocalAdvanceNoticeStatus, 1168
expiration_time
DDS_InvalidLocalAdvanceNoticeStatus, 1168
expired_dropped_sample_count
DDS_DataReaderCacheStatus, 839
expression_parameters
DDS_ContentFilterProperty_t, 829
Extended Qos Support, 660
extensibility_kind
DDS_TypeCode, 1432

facility
NDDS_Config_LogMessage, 2042
fast_heartbeat_period
DDS_RtpsReliableWriterProtocol_t, 1323
file_name
DDS_PersistentStorageSettings, 1244
Filter Use Cases, 412
filter_class_name
DDS_ContentFilterProperty_t, 829
DDS_TopicQuerySelection, 1407
filter_expression
DDS_ChannelSettings_t, 822
DDS_ContentFilterProperty_t, 829
DDS_LocatorFilter_t, 1185
DDS_TopicQuerySelection, 1407
filter_name
DDS_LocatorFilterQosPolicy, 1187
DDS_MultiChannelQosPolicy, 1215
filter_parameters
DDS_TopicQuerySelection, 1407
filtered_sample_bytes
DDS_DataReaderProtocolStatus, 852
DDS_DataWriterProtocolStatus, 901
filtered_sample_bytes_change
DDS_DataReaderProtocolStatus, 852
DDS_DataWriterProtocolStatus, 901
filtered_sample_count
DDS_DataReaderProtocolStatus, 851
DDS_DataWriterProtocolStatus, 901
filtered_sample_count_change
DDS_DataReaderProtocolStatus, 852
DDS_DataWriterProtocolStatus, 901
finalize
DDSContentFilter, 1550
finalize_data
FooTypeSupport, 1987
finalize_data_ex
FooTypeSupport, 1987
finalize_instance
DDSDomainParticipantFactory, 1697
NDDSConfigLogger, 2094
find_member_by_id
DDS_TypeCode, 1456
find_member_by_name
DDS_TypeCode, 1439
find_topic
DDSDomainParticipant, 1657
finish
MyFlatMutableBuilder, 2021
MyFlatUnionBuilder, 2033
rti::flat::FinalSequenceBuilder< ElementOffset >, 1915
rti::flat::MutableArrayBuilder< ElementBuilder, N >, 2009
rti::flat::MutableSequenceBuilder< ElementBuilder >, 2013
rti::flat::PrimitiveSequenceBuilder< T >, 2138
rti::flat::StringBuilder, 2215
finish_sample
MyFlatMutableBuilder, 2021
MyFlatUnionBuilder, 2033
first_available_sample_sequence_number
DDS_DataReaderProtocolStatus, 856
DDS_DataWriterProtocolStatus, 906
first_available_sample_virtual_sequence_number
DDS_DataWriterProtocolStatus, 906

first_unacknowledged_sample_sequence_number
DDS_DataWriterProtocolStatus, 906
first_unacknowledged_sample_subscription_handle
DDS_DataWriterProtocolStatus, 907
first_unacknowledged_sample_virtual_sequence_number
DDS_DataWriterProtocolStatus, 907
first_unelapsed_keep_duration_sample_sequence_number
DDS_DataWriterProtocolStatus, 907
flag
DDS_SampleInfo, 1353
DDS_WriteParams_t, 1522
FlatData Builders, 766
build_data, 768
discard_builder, 768
FlatData Offsets, 771
plain_cast, 773, 775
FlatData Samples, 769
MyFlatFinal, 770
MyFlatMutable, 770
MyFlatUnion, 770
FlatData Topic-Types, 776
FlatSample.hpp, 2300
FlatSampleImpl.hpp, 2303
FlatTypeTraits.hpp, 2304
Flow Control, 282
Flow Controllers, 122
DDS_DEFAULT_FLOW_CONTROLLER_NAME, 125
DDS_EDF_FLOW_CONTROLLER_SCHED_POLICY, 124
DDS_FIXED_RATE_FLOW_CONTROLLER_NAME, 126
DDS_FlowControllerSchedulingPolicy, 123
DDS_HPF_FLOW_CONTROLLER_SCHED_POLICY, 125
DDS_ON_DEMAND_FLOW_CONTROLLER_NAME, 127
DDS_RR_FLOW_CONTROLLER_SCHED_POLICY, 124
flow_controller_allocation
DDS_DomainParticipantResourceLimitsQosPolicy, 986
flow_controller_hash_buckets
DDS_DomainParticipantResourceLimitsQosPolicy, 990
flow_controller_name
DDS_PublishModeQosPolicy, 1281
FlowController Use Cases, 397
flush
DDSDataWriter, 1607
Foo, 1916
FooDataReader, 1917
get_key_value, 1941
is_data_consistent, 1943
lookup_instance, 1942
narrow, 1919
read, 1919
read_instance, 1929
read_instance_w_condition, 1932
read_next_instance, 1935
read_next_instance_w_condition, 1938
read_next_sample, 1927
read_w_condition, 1925
return_loan, 1940
take, 1920
take_instance, 1931
take_instance_w_condition, 1933
take_next_instance, 1936
take_next_instance_w_condition, 1939
take_next_sample, 1928
take_w_condition, 1926
FooDataWriter, 1944
create_data, 1962
delete_data, 1962
discard_loan, 1964
dispose, 1957
dispose_w_params, 1959
dispose_w_timestamp, 1958
get_key_value, 1959
get_loan, 1963
lookup_instance, 1960
narrow, 1946
register_instance, 1946
register_instance_w_params, 1948
register_instance_w_timestamp, 1947
unregister_instance, 1948
unregister_instance_w_params, 1951
unregister_instance_w_timestamp, 1950
write, 1951
write_w_params, 1956
write_w_timestamp, 1955
FooSeq, 1965
∼FooSeq, 1966
copy_no_alloc, 1970
ensure_length, 1974
FooSeq, 1967, 1968
from_array, 1971
get_contiguous_buffer, 1978
get_discontiguous_buffer, 1978
has_ownership, 1979
length, 1973
loan_contiguous, 1976
loan_discontiguous, 1977
maximum, 1975
operator=, 1968
operator[], 1972
to_array, 1971
unloan, 1977
FooTypeSupport, 1979

copy_data, 1984
create_data, 1983
create_data_ex, 1983
data_to_string, 1991
delete_data, 1984
delete_data_ex, 1985
deserialize_data_from_cdr_buffer, 1990
finalize_data, 1987
finalize_data_ex, 1987
get_type_name, 1988
get_typecode, 1991
initialize_data, 1986
initialize_data_ex, 1986
print_data, 1988
register_type, 1981
serialize_data_to_cdr_buffer, 1989
serialize_data_to_cdr_buffer_ex, 1990
unregister_type, 1982
force_interface_poll_detection
NDDS_Transport_UDPv4_Property_t, 2064
NDDS_Transport_UDPv4_WAN_Property_t, 2073
NDDS_Transport_UDPv6_Property_t, 2084
force_type_validation
DDS_TypeConsistencyEnforcementQosPolicy, 1496
Fragmentation, 284
frame_queue_size
NDDS_Utility_NetworkCaptureParams_t, 2089
from_array
FooSeq, 1971
from_cdr_buffer
DDS_DynamicData, 1037
from_micros
Time Support, 519
from_millis
Time Support, 519
from_nanos
Time Support, 519, 520
from_seconds
Time Support, 519, 520
from_string
DDS_DynamicData, 1039
full_reliable_writer_cache
DDS_ReliableWriterCacheChangedStatus, 1303
function_name
DDS_TransportMulticastMappingFunction_t, 1413
Functional Categories, 252

gather_send_buffer_count_max
NDDS_Transport_Property_t, 2047
General Utilities and Compliance Configuration, 190
generation_rank
DDS_SampleInfo, 1350
get
rti::flat::PrimitiveConstOffset< T >, 2134
get_all_datareaders
DDSSubscriber, 1875
get_all_datawriters
DDSPublisher, 1830
get_api_build_version
connext::MessagingVersion, 2007
get_api_version
connext::MessagingVersion, 2007
get_api_version_string
connext::MessagingVersion, 2007
get_boolean
DDS_DynamicData, 1056
get_boolean_array
DDS_DynamicData, 1073
get_boolean_seq
DDS_DynamicData, 1086
get_buffer
rti::flat::OffsetBase, 2128
get_buffer_size
rti::flat::OffsetBase, 2129
get_builtin_subscriber
DDSDomainParticipant, 1661
get_builtin_transport_property
NDDSTransportSupport, 2108
get_c_api_version
NDDSConfigVersion, 2102
get_cdr_buffer
DDS_DynamicData, 1035
get_cdr_serialized_sample_max_size
DDS_TypeCode, 1457
get_char
DDS_DynamicData, 1057
get_char_array
DDS_DynamicData, 1074
get_char_seq
DDS_DynamicData, 1087
get_complex_member
DDS_DynamicData, 1066
get_conditions
DDSAsyncWaitSet, 1540
DDSWaitSet, 1903
get_contiguous_buffer
FooSeq, 1978
get_core_version
NDDSConfigVersion, 2102
get_cpp_api_version
NDDSConfigVersion, 2102
get_current_time
DDSDomainParticipant, 1666
get_data_type
DDSDynamicDataTypeSupport, 1728
get_datareader
DDSReadCondition, 1844
get_datareader_cache_status

DDSDataReader, 1573
get_datareader_protocol_status
DDSDataReader, 1573
get_datareader_qos_from_profile
DDSDomainParticipantFactory, 1707
get_datareader_qos_from_profile_w_topic_name
DDSDomainParticipantFactory, 1708
get_datareaders
DDSSubscriber, 1874
get_datawriter_cache_status
DDSDataWriter, 1594
get_datawriter_protocol_status
DDSDataWriter, 1595
get_datawriter_qos_from_profile
DDSDomainParticipantFactory, 1706
get_datawriter_qos_from_profile_w_topic_name
DDSDomainParticipantFactory, 1706
get_default_datareader_qos
DDSDomainParticipant, 1628
DDSSubscriber, 1863
get_default_datawriter_qos
DDSDomainParticipant, 1626
DDSPublisher, 1821
get_default_flowcontroller_property
DDSDomainParticipant, 1630
get_default_library
DDSDomainParticipant, 1634
DDSDomainParticipantFactory, 1701
DDSPublisher, 1824
DDSSubscriber, 1866
get_default_participant_qos
DDSDomainParticipantFactory, 1700
get_default_profile
DDSDomainParticipant, 1634
DDSDomainParticipantFactory, 1703
DDSPublisher, 1825
DDSSubscriber, 1867
get_default_profile_library
DDSDomainParticipant, 1634
DDSDomainParticipantFactory, 1703
DDSPublisher, 1826
DDSSubscriber, 1867
get_default_publisher_qos
DDSDomainParticipant, 1639
get_default_subscriber_qos
DDSDomainParticipant, 1641
get_default_topic_qos
DDSDomainParticipant, 1636
get_discontiguous_buffer
FooSeq, 1978
get_discovered_participant_data
DDSDomainParticipant, 1671
get_discovered_participant_subject_name
DDSDomainParticipant, 1672
get_discovered_participants
DDSDomainParticipant, 1670
get_discovered_participants_from_subject_name
DDSDomainParticipant, 1670
get_discovered_topic_data
DDSDomainParticipant, 1673
get_discovered_topics
DDSDomainParticipant, 1673
get_discriminator
DDS_DynamicData, 1067
get_dns_tracker_polling_period
DDSDomainParticipant, 1680
get_domain_id
DDSDomainParticipant, 1665
get_double
DDS_DynamicData, 1056
get_double_array
DDS_DynamicData, 1072
get_double_seq
DDS_DynamicData, 1085
get_element
rti::flat::AbstractPrimitiveList< T >, 791
rti::flat::FinalAlignedArrayOffset< ElementOffset, N >, 1911
rti::flat::FinalArrayOffset< ElementOffset, N >, 1912
rti::flat::MutableArrayOffset< ElementOffset, N >, 2011
rti::flat::SequenceOffset< ElementOffset >, 2205
get_enabled_statuses
DDSStatusCondition, 1846
get_entity
DDSStatusCondition, 1847
get_expression_parameters
DDSContentFilteredTopic, 1553
DDSMultiTopic, 1799
get_filter_expression
DDSContentFilteredTopic, 1552
get_float
DDS_DynamicData, 1055
get_float_array
DDS_DynamicData, 1071
get_float_seq
DDS_DynamicData, 1084
get_guid
DDSTopicQuery, 1895
get_handler
DDSCondition, 1546
get_implicit_publisher
DDSDomainParticipant, 1682
get_implicit_subscriber
DDSDomainParticipant, 1683
get_inconsistent_topic_status
DDSTopic, 1887
get_info

DDS_DynamicData, 1040
get_instance
DDS_TypeCodeFactory, 1479
DDSDomainParticipantFactory, 1697
NDDSConfigLogger, 2094
NDDSConfigVersion, 2102
get_instance_handle
DDSDataReader, 1582
DDSDataWriter, 1611
DDSEntity, 1736
get_instance_state_mask
DDSReadCondition, 1844
get_int8
DDS_DynamicData, 1064
get_int8_array
DDS_DynamicData, 1079
get_int8_seq
DDS_DynamicData, 1092
get_key_value
DDSKeyedOctetsDataReader, 1748, 1749
DDSKeyedOctetsDataWriter, 1762
DDSKeyedStringDataReader, 1776, 1777
DDSKeyedStringDataWriter, 1787
FooDataReader, 1941
FooDataWriter, 1959
get_listener
DDSDataReader, 1577
DDSDataWriter, 1607
DDSDomainParticipant, 1682
DDSPublisher, 1838
DDSSubscriber, 1880
DDSTopic, 1890
get_liveliness_changed_status
DDSDataReader, 1570
get_liveliness_lost_status
DDSDataWriter, 1592
get_loan
FooDataWriter, 1963
get_long
DDS_DynamicData, 1051
get_long_array
DDS_DynamicData, 1067
get_long_seq
DDS_DynamicData, 1081
get_longdouble
DDS_DynamicData, 1060
get_longdouble_array
DDS_DynamicData, 1077
get_longdouble_seq
DDS_DynamicData, 1090
get_longlong
DDS_DynamicData, 1059
get_longlong_array
DDS_DynamicData, 1075
get_longlong_seq
DDS_DynamicData, 1088
get_matched_publication_data
DDSDataReader, 1568
get_matched_publication_datareader_protocol_status
DDSDataReader, 1573
get_matched_publication_participant_data
DDSDataReader, 1569
get_matched_publications
DDSDataReader, 1567
get_matched_subscription_data
DDSDataWriter, 1600
get_matched_subscription_datawriter_protocol_status
DDSDataWriter, 1595
get_matched_subscription_datawriter_protocol_status_by_locator
DDSDataWriter, 1597
get_matched_subscription_locators
DDSDataWriter, 1598
get_matched_subscription_participant_data
DDSDataWriter, 1601
get_matched_subscriptions
DDSDataWriter, 1599
get_member_count
DDS_DynamicData, 1045
get_member_info
DDS_DynamicData, 1047
get_member_info_by_index
DDS_DynamicData, 1049
get_member_type
DDS_DynamicData, 1050
get_name
DDSFlowController, 1739
DDSTopicDescription, 1893
get_number_of_properties
PROPERTY, 621
get_number_of_tags
DATA_TAG, 576
get_octet
DDS_DynamicData, 1058
get_octet_array
DDS_DynamicData, 1075
get_octet_seq
DDS_DynamicData, 1087
get_offered_deadline_missed_status
DDSDataWriter, 1592
get_offered_incompatible_qos_status
DDSDataWriter, 1593
get_output_device
NDDSConfigLogger, 2096
get_output_file
NDDSConfigLogger, 2096
get_participant
DDSFlowController, 1739
DDSPublisher, 1833

DDSSubscriber, 1876
DDSTopicDescription, 1893
get_participant_factory_qos_from_profile
DDSDomainParticipantFactory, 1703
get_participant_protocol_status
DDSDomainParticipant, 1676
get_participant_qos_from_profile
DDSDomainParticipantFactory, 1704
get_participants
DDSDomainParticipantFactory, 1721
get_primitive_tc
DDS_TypeCodeFactory, 1480
get_print_format
NDDSConfigLogger, 2097
get_print_format_by_log_level
NDDSConfigLogger, 2098
get_product_version
NDDSConfigVersion, 2102
get_properties
PROPERTY, 626
get_property
DDSAsyncWaitSet, 1539
DDSFlowController, 1738
DDSWaitSet, 1904
get_publication_matched_status
DDSDataWriter, 1593
get_publisher
DDSDataWriter, 1602
get_publisher_qos_from_profile
DDSDomainParticipantFactory, 1705
get_publishers
DDSDomainParticipant, 1649
get_qos
DDSDataReader, 1575
DDSDataWriter, 1606
DDSDomainParticipant, 1678
DDSDomainParticipantFactory, 1715
DDSPublisher, 1837
DDSSubscriber, 1879
DDSTopic, 1889
get_qos_profile_libraries
DDSDomainParticipantFactory, 1710
get_qos_profiles
DDSDomainParticipantFactory, 1710
get_query_expression
DDSQueryCondition, 1842
get_query_parameters
DDSQueryCondition, 1842
get_related_topic
DDSContentFilteredTopic, 1556
get_reliable_reader_activity_changed_status
DDSDataWriter, 1594
get_reliable_writer_cache_changed_status
DDSDataWriter, 1594
get_reply_datareader
connext::Requester< TReq, TRep >, 2177
get_reply_datawriter
connext::Replier< TReq, TRep >, 2151
get_request_datareader
connext::Replier< TReq, TRep >, 2150
get_request_datawriter
connext::Requester< TReq, TRep >, 2177
get_requested_deadline_missed_status
DDSDataReader, 1571
get_requested_incompatible_qos_status
DDSDataReader, 1571
get_rpc_mask
NDDSConfigCompliance, 2091
get_sample_lost_status
DDSDataReader, 1572
get_sample_rejected_status
DDSDataReader, 1570
get_sample_state_mask
DDSReadCondition, 1843
get_service_request_accepted_status
DDSDataWriter, 1596
get_short
DDS_DynamicData, 1052
get_short_array
DDS_DynamicData, 1068
get_short_seq
DDS_DynamicData, 1082
get_spin_per_microsecond
NDDSUtility, 2112
get_status_changes
DDSDataReader, 1582
DDSDataWriter, 1611
DDSEntity, 1735
get_statuscondition
DDSDataReader, 1581
DDSDataWriter, 1611
DDSEntity, 1735
get_stream_kind_mask
DDSReadCondition, 1844
get_string
DDS_DynamicData, 1062
rti::flat::StringOffset, 2216
get_subscriber
DDSDataReader, 1570
get_subscriber_qos_from_profile
DDSDomainParticipantFactory, 1705
get_subscribers
DDSDomainParticipant, 1650
get_subscription_expression
DDSMultiTopic, 1799
get_subscription_matched_status
DDSDataReader, 1572
get_topic

DDSDataWriter, 1601
get_topic_qos_from_profile
DDSDomainParticipantFactory, 1708
get_topic_qos_from_profile_w_topic_name
DDSDomainParticipantFactory, 1709
get_topicdescription
DDSDataReader, 1569
get_transport_plugin
NDDSTransportSupport, 2110
get_trigger_value
DDSCondition, 1545
DDSGuardCondition, 1741
get_type
DDS_DynamicData, 1045
get_type_kind
DDS_DynamicData, 1045
get_type_name
DDSDynamicDataTypeSupport, 1728
DDSKeyedOctetsTypeSupport, 1767
DDSKeyedStringTypeSupport, 1792
DDSOctetsTypeSupport, 1813
DDSStringTypeSupport, 1857
DDSTopicDescription, 1892
FooTypeSupport, 1988
get_type_object_serialized_size
DDS_TypeCode, 1457
get_typecode
DDSDomainParticipant, 1675
DDSKeyedOctetsTypeSupport, 1768
DDSKeyedStringTypeSupport, 1793
DDSOctetsTypeSupport, 1814
DDSStringTypeSupport, 1858
FooTypeSupport, 1991
get_typecode_from_config
DDSDomainParticipantFactory, 1717
get_uint8
DDS_DynamicData, 1065
get_uint8_array
DDS_DynamicData, 1080
get_uint8_seq
DDS_DynamicData, 1092
get_ulong
DDS_DynamicData, 1053
get_ulong_array
DDS_DynamicData, 1069
get_ulong_seq
DDS_DynamicData, 1082
get_ulonglong
DDS_DynamicData, 1060
get_ulonglong_array
DDS_DynamicData, 1076
get_ulonglong_seq
DDS_DynamicData, 1089
get_ushort
DDS_DynamicData, 1054
get_ushort_array
DDS_DynamicData, 1070
get_ushort_seq
DDS_DynamicData, 1083
get_verbosity
NDDSConfigLogger, 2095
get_verbosity_by_category
NDDSConfigLogger, 2095
get_view_state_mask
DDSReadCondition, 1844
get_wchar
DDS_DynamicData, 1061
get_wchar_array
DDS_DynamicData, 1078
get_wchar_seq
DDS_DynamicData, 1091
get_wstring
DDS_DynamicData, 1063
get_xtypes_mask
NDDSConfigCompliance, 2090
group_coherent_set_sequence_number
DDS_CoherentSetInfo_t, 825
GROUP_DATA, 604
DDS_GROUPDATA_QOS_POLICY_NAME, 605
group_data
DDS_PublicationBuiltinTopicData, 1269
DDS_PublisherQos, 1278
DDS_SubscriberQos, 1369
DDS_SubscriptionBuiltinTopicData, 1375
group_guid
DDS_CoherentSetInfo_t, 825
GUID Support, 524
DDS_GUID_AUTO, 527
DDS_GUID_compare, 526
DDS_GUID_copy, 526
DDS_GUID_equals, 525
DDS_GUID_t, 525
DDS_GUID_UNKNOWN, 527
DDS_GUID_ZERO, 527
DDS_RTPS_EntityId_t, 525
DDS_RTPS_GUID_t, 525
DDS_RTPS_GuidPrefix_t, 525

handle
DDS_WriteParams_t, 1521
has_ownership
FooSeq, 1979
Heap Monitoring, 746
NDDS_UTILITY_HEAP_MONITORING_SNAPSHOT_CONTENT_BIT_
747
NDDS_UTILITY_HEAP_MONITORING_SNAPSHOT_CONTENT_BIT_
747

NDDS_UTILITY_HEAP_MONITORING_SNAPSHOT_CONTENT_BITFIELD, 747
NDDS_UTILITY_HEAP_MONITORING_SNAPSHOT_CONTENT_DEFAULT, 747
NDDS_UTILITY_HEAP_MONITORING_SNAPSHOT_CONTENT_MAX, 747
NDDS_Utility_HeapMonitoringSnapshotContentFormat, 746
Heap Support in C, 677
DDS_Heap_calloc, 678
DDS_Heap_free, 678
DDS_Heap_malloc, 678
heartbeat_period
DDS_RtpsReliableWriterProtocol_t, 1323
heartbeat_suppression_duration
DDS_RtpsReliableReaderProtocol_t, 1316
heartbeats_per_max_samples
DDS_RtpsReliableWriterProtocol_t, 1327
high
DDS_SequenceNumber_t, 1358
high_watermark
DDS_RtpsReliableWriterProtocol_t, 1322
high_watermark_reliable_writer_cache
DDS_ReliableWriterCacheChangedStatus, 1304
HISTORY, 603
DDS_HISTORY_QOS_POLICY_NAME, 604
DDS_HistoryQosPolicyKind, 603
DDS_KEEP_ALL_HISTORY_QOS, 604
DDS_KEEP_LAST_HISTORY_QOS, 604
history
DDS_DataReaderQos, 862
DDS_DataWriterQos, 913
DDS_TopicBuiltinTopicData, 1395
DDS_TopicQos, 1401
history_depth
DDS_DurabilityServiceQosPolicy, 1011
history_kind
DDS_DurabilityServiceQosPolicy, 1010
host_port
NDDS_Transport_UDP_WAN_CommPortsMappingInfo, 2056
howto_prototypes.h, 2227

id
DDS_StructMember, 1366
DDS_ValueMember, 1507
identity
connext::Sample< T >, 2185
connext::SampleRef< T >, 2197
connext::WriteSample< T >, 2222
connext::WriteSampleRef< T >, 2226
DDS_WriteParams_t, 1520
ignore_default_domain_announcements
DDS_DiscoveryConfigQosPolicy, 950
ignore_enum_literal_names
DDS_TypeConsistencyEnforcementQosPolicy, 1496
ignore_environment_profile
DDS_ProfileQosPolicy, 1258
ignore_loopback_interface
NDDS_Transport_UDPv4_Property_t, 2060
NDDS_Transport_UDPv4_WAN_Property_t, 2069
NDDS_Transport_UDPv6_Property_t, 2080
ignore_member_names
DDS_TypeConsistencyEnforcementQosPolicy, 1495
ignore_nonrunning_interfaces
NDDS_Transport_UDPv4_Property_t, 2061
NDDS_Transport_UDPv4_WAN_Property_t, 2070
NDDS_Transport_UDPv6_Property_t, 2081
ignore_nonup_interfaces
NDDS_Transport_UDPv4_Property_t, 2060
NDDS_Transport_UDPv4_WAN_Property_t, 2069
ignore_participant
DDSDomainParticipant, 1661
ignore_publication
DDSDomainParticipant, 1664
ignore_resource_profile
DDS_ProfileQosPolicy, 1259
ignore_sequence_bounds
DDS_TypeConsistencyEnforcementQosPolicy, 1495
ignore_string_bounds
DDS_TypeConsistencyEnforcementQosPolicy, 1495
ignore_subscription
DDSDomainParticipant, 1664
ignore_topic
DDSDomainParticipant, 1663
ignore_user_profile
DDS_ProfileQosPolicy, 1258
ignored_entity_allocation
DDS_DomainParticipantResourceLimitsQosPolicy, 984
ignored_entity_hash_buckets
DDS_DomainParticipantResourceLimitsQosPolicy, 989
ignored_entity_replacement_kind
DDS_DomainParticipantResourceLimitsQosPolicy, 1001
Immutable QoS, 333
inactivate_nonprogressing_readers
DDS_RtpsReliableWriterProtocol_t, 1326
inactive_count
DDS_ReliableReaderActivityChangedStatus, 1301
inactive_count_change
DDS_ReliableReaderActivityChangedStatus, 1302
include_root_elements
DDS_PrintFormatProperty, 1255
incomplete_coherent_set
DDS_CoherentSetInfo_t, 826
incremental_count

DDS_AllocationSettings_t, 797
indent
DDS_QosPrintFormat, 1286
DDS_TypeCodePrintFormatProperty, 1490
info
connext::Sample< T >, 2186
connext::SampleRef< T >, 2195
connext::WriteSample< T >, 2222
connext::WriteSampleRef< T >, 2225
informational
NDDSConfigLogger, 2099
Infrastructure, 192
make_valid_sample_iterator, 194
move, 195
Infrastructure Module, 167
DDS_QOS_PRINT_ALL, 169
initial_active_topic_queries
DDS_DataWriterResourceLimitsQosPolicy, 924
initial_batches
DDS_DataWriterResourceLimitsQosPolicy, 920
initial_concurrent_blocking_threads
DDS_DataWriterResourceLimitsQosPolicy, 919
initial_count
DDS_AllocationSettings_t, 797
DDS_EventQosPolicy, 1151
initial_fragmented_samples
DDS_BuiltinTopicReaderResourceLimits_t, 819
DDS_DataReaderResourceLimitsQosPolicy, 875
initial_infos
DDS_BuiltinTopicReaderResourceLimits_t, 817
DDS_DataReaderResourceLimitsQosPolicy, 872
initial_instances
DDS_ResourceLimitsQosPolicy, 1312
initial_objects_per_thread
DDS_SystemResourceLimitsQosPolicy, 1383
initial_outstanding_reads
DDS_BuiltinTopicReaderResourceLimits_t, 817
DDS_DataReaderResourceLimitsQosPolicy, 873
initial_participant_announcements
DDS_DiscoveryConfigQosPolicy, 939
initial_peers
DDS_DiscoveryQosPolicy, 962
initial_records
DDS_DatabaseQosPolicy, 833
initial_remote_readers
DDS_DataWriterResourceLimitsQosPolicy, 926
initial_remote_virtual_writers
DDS_DataReaderResourceLimitsQosPolicy, 878
initial_remote_virtual_writers_per_instance
DDS_DataReaderResourceLimitsQosPolicy, 879
initial_remote_writers
DDS_DataReaderResourceLimitsQosPolicy, 871
initial_remote_writers_per_instance
DDS_DataReaderResourceLimitsQosPolicy, 872
initial_samples
DDS_BuiltinTopicReaderResourceLimits_t, 816
DDS_ResourceLimitsQosPolicy, 1312
initial_topic_queries
DDS_DataReaderResourceLimitsQosPolicy, 882
initial_virtual_sequence_number
DDS_DataWriterProtocolQosPolicy, 897
initial_virtual_writers
DDS_DataWriterResourceLimitsQosPolicy, 923
initial_weak_references
DDS_DatabaseQosPolicy, 835
initialize_data
FooTypeSupport, 1986
initialize_data_ex
FooTypeSupport, 1986
initialize_writer_loaned_sample
DDS_DataWriterResourceLimitsQosPolicy, 925
Installing Transport Plugins, 177
NDDS_Transport_create_plugin, 181
NDDS_Transport_Handle_is_nil, 182
NDDS_TRANSPORT_HANDLE_NIL, 182
NDDS_Transport_Handle_t, 181
Instance Lifecycle, 286
Instance States, 164
DDS_ALIVE_INSTANCE_STATE, 166
DDS_ANY_INSTANCE_STATE, 166
DDS_InstanceStateKind, 165
DDS_InstanceStateMask, 165
DDS_NOT_ALIVE_DISPOSED_INSTANCE_STATE, 166
DDS_NOT_ALIVE_INSTANCE_STATE, 166
DDS_NOT_ALIVE_NO_WRITERS_INSTANCE_STATE, 166
instance_handle
DDS_SampleInfo, 1349
instance_hash_buckets
DDS_ResourceLimitsQosPolicy, 1313
instance_id
DDS_ServiceRequest, 1360
instance_replacement
DDS_DataReaderResourceLimitsQosPolicy, 883
DDS_DataWriterResourceLimitsQosPolicy, 921
instance_state
DDS_SampleInfo, 1348
instance_state_consistency_kind
DDS_ReliabilityQosPolicy, 1299
instance_states
DDS_ReadConditionParams, 1289
interceptor
DDS_EndpointTrustAlgorithmInfo, 1143
DDS_ParticipantTrustAlgorithmInfo, 1236
Interface, 440
NDDS_TRANSPORT_INTERFACE_OFF, 441
NDDS_TRANSPORT_INTERFACE_ON, 441

NDDS_Transport_Interface_Status_t, 440
interface_poll_period
NDDS_Transport_UDPv4_Property_t, 2064
NDDS_Transport_UDPv4_WAN_Property_t, 2073
NDDS_Transport_UDPv6_Property_t, 2084
Interpreter.hpp, 2334
is_alias_pointer
DDS_TypeCode, 1452
is_assignable_from
DDS_TypeCode, 1436
is_cdr
DDS_DynamicData, 1035
is_cpp_compatible
rti::flat::OffsetBase, 2128
is_data_consistent
FooDataReader, 1943
is_key
DDS_StructMember, 1366
DDS_ValueMember, 1507
is_license_message
NDDS_Config_LogMessage, 2041
is_matched_publication_alive
DDSDataReader, 1568
is_matched_subscription_active
DDSDataWriter, 1596
is_member_bitfield
DDS_TypeCode, 1445
is_member_key
DDS_DynamicData, 1051
DDS_TypeCode, 1443
is_member_must_understand
DDS_TypeCode, 1473
is_member_pointer
DDS_TypeCode, 1444
is_member_required
DDS_TypeCode, 1443
is_must_understand
DDS_StructMember, 1366
DDS_ValueMember, 1508
is_nested
rti::flat::AbstractBuilder, 788
is_nil_data
connext::SampleRef< T >, 2196
connext::WriteSampleRef< T >, 2226
is_nil_info
connext::SampleRef< T >, 2197
is_null
rti::flat::OffsetBase, 2128
rti::flat::SequenceIterator< E, OffsetKind >, 2201
is_optional
DDS_StructMember, 1366
DDS_ValueMember, 1507
is_pointer
DDS_StructMember, 1366
DDS_UnionMember, 1502
DDS_ValueMember, 1507
is_sample_app_acknowledged
DDSDataWriter, 1603
is_security_message
NDDS_Config_LogMessage, 2041
is_standalone
DDS_QosPrintFormat, 1285
is_valid
DDS_DynamicData, 1030
DDSDynamicDataTypeSupport, 1727
rti::flat::AbstractBuilder, 788
IsReplyRelatedPredicate
connext::IsReplyRelatedPredicate< T >, 1994
iterator
connext::LoanedSamples< T >, 1997
rti::flat::AbstractAlignedList< ElementOffset >, 786
iterator_category
rti::flat::SequenceIterator< E, OffsetKind >, 2200

join_multicast_group_timeout
NDDS_Transport_UDPv4_Property_t, 2065
NDDS_Transport_UDPv6_Property_t, 2085
journal_kind
DDS_PersistentStorageSettings, 1245

keep_minimum_state_for_instances
DDS_DataReaderResourceLimitsQosPolicy, 881
key
DDS_KeyedOctets, 1170
DDS_KeyedString, 1172
DDS_ParticipantBuiltinTopicData, 1230
DDS_PublicationBuiltinTopicData, 1266
DDS_SubscriptionBuiltinTopicData, 1372
DDS_TopicBuiltinTopicData, 1392
key_establishment
DDS_ParticipantTrustAlgorithmInfo, 1236
key_only_filter
DDS_ExpressionProperty, 1153
KeyedOctets Built-in Type, 511
DDS_KeyedOctets, 511
KeyedString Built-in Type, 509
DDS_KeyedString, 509
kind
DDS_DestinationOrderQosPolicy, 933
DDS_DurabilityQosPolicy, 1007
DDS_HistoryQosPolicy, 1164
DDS_LivelinessQosPolicy, 1182
DDS_Locator_t, 1184
DDS_OwnershipQosPolicy, 1227
DDS_PrintFormatProperty, 1254
DDS_PublishModeQosPolicy, 1281
DDS_ReliabilityQosPolicy, 1298
DDS_ServiceQosPolicy, 1359
DDS_TopicQuerySelection, 1407

DDS_TransportMulticastQosPolicy, 1416
DDS_TypeCode, 1432
DDS_TypeConsistencyEnforcementQosPolicy, 1494

labels
DDS_UnionMember, 1502
Large Data Use Cases, 418
last_available_sample_sequence_number
DDS_DataReaderProtocolStatus, 856
DDS_DataWriterProtocolStatus, 906
last_available_sample_virtual_sequence_number
DDS_DataWriterProtocolStatus, 906
last_committed_sample_sequence_number
DDS_DataReaderProtocolStatus, 856
last_corrupted_message_timestamp
DDS_DomainParticipantProtocolStatus, 970
last_instance_handle
DDS_OfferedDeadlineMissedStatus, 1220
DDS_ReliableReaderActivityChangedStatus, 1302
DDS_RequestedDeadlineMissedStatus, 1306
DDS_SampleRejectedStatus, 1357
last_policy_id
DDS_OfferedIncompatibleQosStatus, 1222
DDS_RequestedIncompatibleQosStatus, 1308
last_publication_handle
DDS_LivelinessChangedStatus, 1178
DDS_SubscriptionMatchedStatus, 1381
last_reason
DDS_SampleLostStatus, 1356
DDS_SampleRejectedStatus, 1357
last_request_handle
DDS_ServiceRequestAcceptedStatus, 1362
last_subscription_handle
DDS_PublicationMatchedStatus, 1275
late_joiner_heartbeat_period
DDS_RtpsReliableWriterProtocol_t, 1324
LATENCY_BUDGET, 605
DDS_LATENCYBUDGET_QOS_POLICY_NAME, 606
latency_budget
DDS_DataReaderQos, 861
DDS_DataWriterQos, 912
DDS_PublicationBuiltinTopicData, 1267
DDS_SubscriptionBuiltinTopicData, 1373
DDS_TopicBuiltinTopicData, 1394
DDS_TopicQos, 1401
lazily_allocate_batch_resources
DDS_DataReaderResourceLimitsQosPolicy, 885
lease_duration
DDS_LivelinessQosPolicy, 1182
length
connext::LoanedSamples< T >, 2002
DDS_KeyedOctets, 1170
DDS_Octets, 1218
DDS_TypeCode, 1448
FooSeq, 1973
level
DDS_AsyncWaitSetProperty_t, 805
NDDS_Config_LogMessage, 2041
LIFESPAN, 606
DDS_LIFESPAN_QOS_POLICY_NAME, 606
lifespan
DDS_DataWriterQos, 914
DDS_PublicationBuiltinTopicData, 1267
DDS_TopicBuiltinTopicData, 1394
DDS_TopicQos, 1402
LIVELINESS, 607
DDS_AUTOMATIC_LIVELINESS_QOS, 608
DDS_LIVELINESS_QOS_POLICY_NAME, 608
DDS_LivelinessQosPolicyKind, 607
DDS_MANUAL_BY_PARTICIPANT_LIVELINESS_QOS, 608
DDS_MANUAL_BY_TOPIC_LIVELINESS_QOS, 608
Liveliness, 292
liveliness
DDS_DataReaderQos, 862
DDS_DataWriterQos, 913
DDS_PublicationBuiltinTopicData, 1267
DDS_SubscriptionBuiltinTopicData, 1373
DDS_TopicBuiltinTopicData, 1394
DDS_TopicQos, 1401
load_compliance_masks
NDDSConfigCompliance, 2091
load_profiles
DDSDomainParticipantFactory, 1715
loan_contiguous
FooSeq, 1976
loan_discontiguous
FooSeq, 1977
LoanedSamples
connext::LoanedSamples< T >, 1998
local_publisher_allocation
DDS_DomainParticipantResourceLimitsQosPolicy, 981
local_publisher_hash_buckets
DDS_DomainParticipantResourceLimitsQosPolicy, 987
local_reader_allocation
DDS_DomainParticipantResourceLimitsQosPolicy, 981
local_reader_hash_buckets
DDS_DomainParticipantResourceLimitsQosPolicy, 986
local_subscriber_allocation
DDS_DomainParticipantResourceLimitsQosPolicy, 981
local_subscriber_hash_buckets

DDS_DomainParticipantResourceLimitsQosPolicy, 987
local_topic_allocation
DDS_DomainParticipantResourceLimitsQosPolicy, 982
local_topic_hash_buckets
DDS_DomainParticipantResourceLimitsQosPolicy, 987
local_writer_allocation
DDS_DomainParticipantResourceLimitsQosPolicy, 981
local_writer_hash_buckets
DDS_DomainParticipantResourceLimitsQosPolicy, 986
locator_filter
DDS_PublicationBuiltinTopicData, 1271
locator_filters
DDS_LocatorFilterQosPolicy, 1187
locator_reachability_assert_period
DDS_DiscoveryConfigQosPolicy, 952
locator_reachability_change_detection_period
DDS_DiscoveryConfigQosPolicy, 953
locator_reachability_lease_duration
DDS_DiscoveryConfigQosPolicy, 953
LOCATORFILTER, 608
DDS_LOCATORFILTER_QOS_POLICY_NAME, 609
locators
DDS_LocatorFilter_t, 1185
LOGGING, 609
Logging, 294, 722
NDDS_CONFIG_LOG_CATEGORY_ALL, 726
NDDS_CONFIG_LOG_CATEGORY_API, 726
NDDS_CONFIG_LOG_CATEGORY_COMMUNICATION, 726
NDDS_CONFIG_LOG_CATEGORY_DATABASE, 726
NDDS_CONFIG_LOG_CATEGORY_DISCOVERY, 726
NDDS_CONFIG_LOG_CATEGORY_ENTITIES, 726
NDDS_CONFIG_LOG_CATEGORY_PLATFORM, 726
NDDS_CONFIG_LOG_CATEGORY_SECURITY, 726
NDDS_CONFIG_LOG_CATEGORY_USER, 726
NDDS_CONFIG_LOG_FACILITY_MIDDLEWARE, 727
NDDS_CONFIG_LOG_FACILITY_SECURITY_EVENT, 727
NDDS_CONFIG_LOG_FACILITY_SERVICE, 727
NDDS_CONFIG_LOG_FACILITY_USER, 727
NDDS_CONFIG_LOG_LEVEL_DEBUG, 724
NDDS_CONFIG_LOG_LEVEL_ERROR, 724
NDDS_CONFIG_LOG_LEVEL_FATAL_ERROR, 724
NDDS_CONFIG_LOG_LEVEL_STATUS_LOCAL, 724
NDDS_CONFIG_LOG_LEVEL_STATUS_REMOTE, 724
NDDS_CONFIG_LOG_LEVEL_WARNING, 724
NDDS_CONFIG_LOG_PRINT_FORMAT_DEBUG, 727
NDDS_CONFIG_LOG_PRINT_FORMAT_DEFAULT, 726
NDDS_CONFIG_LOG_PRINT_FORMAT_MAXIMAL, 727
NDDS_CONFIG_LOG_PRINT_FORMAT_MINIMAL, 727
NDDS_CONFIG_LOG_PRINT_FORMAT_TIMESTAMPED, 726
NDDS_CONFIG_LOG_PRINT_FORMAT_VERBOSE, 727
NDDS_CONFIG_LOG_PRINT_FORMAT_VERBOSE_TIMESTAMPED, 727
NDDS_CONFIG_LOG_VERBOSITY_ERROR, 724
NDDS_CONFIG_LOG_VERBOSITY_SILENT, 724
NDDS_CONFIG_LOG_VERBOSITY_STATUS_ALL, 724
NDDS_CONFIG_LOG_VERBOSITY_STATUS_LOCAL, 724
NDDS_CONFIG_LOG_VERBOSITY_STATUS_REMOTE, 724
NDDS_CONFIG_LOG_VERBOSITY_WARNING, 724
NDDS_Config_LogCategory, 725
NDDS_Config_LogFacility, 727
NDDS_Config_LogLevel, 724
NDDS_Config_LogPrintFormat, 726
NDDS_Config_LogVerbosity, 723
NDDS_CONFIG_SYSLOG_LEVEL_ALERT, 725
NDDS_CONFIG_SYSLOG_LEVEL_CRITICAL, 725
NDDS_CONFIG_SYSLOG_LEVEL_DEBUG, 725
NDDS_CONFIG_SYSLOG_LEVEL_EMERGENCY, 725
NDDS_CONFIG_SYSLOG_LEVEL_ERROR, 725
NDDS_CONFIG_SYSLOG_LEVEL_INFORMATIONAL, 725
NDDS_CONFIG_SYSLOG_LEVEL_NOTICE, 725
NDDS_CONFIG_SYSLOG_LEVEL_WARNING, 725
NDDS_CONFIG_SYSLOG_VERBOSITY_ALERT, 728
NDDS_CONFIG_SYSLOG_VERBOSITY_CRITICAL, 728
NDDS_CONFIG_SYSLOG_VERBOSITY_DEBUG, 728
NDDS_CONFIG_SYSLOG_VERBOSITY_EMERGENCY, 728
NDDS_CONFIG_SYSLOG_VERBOSITY_ERROR, 728
NDDS_CONFIG_SYSLOG_VERBOSITY_INFORMATIONAL,

728
NDDS_CONFIG_SYSLOG_VERBOSITY_NOTICE, 728
NDDS_CONFIG_SYSLOG_VERBOSITY_SILENT, 728
NDDS_CONFIG_SYSLOG_VERBOSITY_WARNING, 728
NDDS_Config_SyslogLevel, 725
NDDS_Config_SyslogVerbosity, 727
logging
DDS_DomainParticipantFactoryQos, 969
Logging and Version, 189
logging_settings
DDS_MonitoringDistributionSettings, 1197
logs
DDS_MonitoringTelemetryData, 1213
lookup_contentfilter
DDSDomainParticipant, 1633
lookup_datareader
DDSSubscriber, 1872
lookup_datareader_by_name
DDSDomainParticipant, 1692
DDSSubscriber, 1880
lookup_datawriter
DDSPublisher, 1829
lookup_datawriter_by_name
DDSDomainParticipant, 1691
DDSPublisher, 1839
lookup_flowcontroller
DDSDomainParticipant, 1660
lookup_instance
DDSKeyedOctetsDataReader, 1749
DDSKeyedOctetsDataWriter, 1762, 1763
DDSKeyedStringDataReader, 1777
DDSKeyedStringDataWriter, 1788
FooDataReader, 1942
FooDataWriter, 1960
lookup_participant
DDSDomainParticipantFactory, 1713
lookup_participant_by_name
DDSDomainParticipantFactory, 1719
lookup_property
PROPERTY, 625
lookup_publisher_by_name
DDSDomainParticipant, 1690
lookup_subscriber_by_name
DDSDomainParticipant, 1690
lookup_tag
DATA_TAG, 577
lookup_topic_query
DDSDataReader, 1578
lookup_topicdescription
DDSDomainParticipant, 1658
lookup_transport
NDDSTransportSupport, 2106
low
DDS_SequenceNumber_t, 1358
low_watermark
DDS_RtpsReliableWriterProtocol_t, 1321
low_watermark_reliable_writer_cache
DDS_ReliableWriterCacheChangedStatus, 1303

major
DDS_ProductVersion_t, 1256
DDS_ProtocolVersion_t, 1263
NDDS_Config_LibraryVersion_t, 2039
major_version
connext::MessagingLibraryVersion, 2006
make_valid_sample_iterator
Infrastructure, 194
mapping_function
DDS_TransportMulticastMapping_t, 1412
mask
DDS_ThreadSettings_t, 1386
DDS_TransportBuiltinQosPolicy, 1408
Matching, 382
matching_reader_writer_pair_allocation
DDS_DomainParticipantResourceLimitsQosPolicy, 983
matching_reader_writer_pair_hash_buckets
DDS_DomainParticipantResourceLimitsQosPolicy, 989
matching_writer_reader_pair_allocation
DDS_DomainParticipantResourceLimitsQosPolicy, 983
matching_writer_reader_pair_hash_buckets
DDS_DomainParticipantResourceLimitsQosPolicy, 989
max_active_topic_queries
DDS_DataWriterResourceLimitsQosPolicy, 924
max_app_ack_remote_readers
DDS_DataWriterResourceLimitsQosPolicy, 924
max_app_ack_response_length
DDS_DataReaderResourceLimitsQosPolicy, 881
max_batches
DDS_DataWriterResourceLimitsQosPolicy, 921
max_blocking_time
DDS_ReliabilityQosPolicy, 1298
max_bytes_per_file
DDS_LoggingQosPolicy, 1190
max_bytes_per_nack_response
DDS_RtpsReliableWriterProtocol_t, 1329
max_concurrent_blocking_threads
DDS_DataWriterResourceLimitsQosPolicy, 919
max_count
DDS_AllocationSettings_t, 797
DDS_EventQosPolicy, 1152
max_data_availability_waiting_time

DDS_AvailabilityQosPolicy, 808
max_data_bytes
DDS_BatchQosPolicy, 811
max_endpoint_availability_waiting_time
DDS_AvailabilityQosPolicy, 808
max_endpoint_group_cumulative_characters
DDS_DomainParticipantResourceLimitsQosPolicy, 1000
max_endpoint_groups
DDS_DomainParticipantResourceLimitsQosPolicy, 1000
max_event_count
DDS_WaitSetProperty_t, 1510
max_event_delay
DDS_WaitSetProperty_t, 1510
max_files
DDS_LoggingQosPolicy, 1191
max_flush_delay
DDS_BatchQosPolicy, 812
max_fragmented_samples
DDS_BuiltinTopicReaderResourceLimits_t, 819
DDS_DataReaderResourceLimitsQosPolicy, 874
max_fragmented_samples_per_remote_writer
DDS_BuiltinTopicReaderResourceLimits_t, 820
DDS_DataReaderResourceLimitsQosPolicy, 875
max_fragments_per_sample
DDS_BuiltinTopicReaderResourceLimits_t, 820
DDS_DataReaderResourceLimitsQosPolicy, 876
max_gather_destinations
DDS_DomainParticipantResourceLimitsQosPolicy, 991
max_heartbeat_response_delay
DDS_RtpsReliableReaderProtocol_t, 1315
max_heartbeat_retries
DDS_RtpsReliableWriterProtocol_t, 1326
max_historical_logs
DDS_MonitoringLoggingDistributionSettings, 1202
max_infos
DDS_BuiltinTopicReaderResourceLimits_t, 817
DDS_DataReaderResourceLimitsQosPolicy, 871
max_initial_participant_announcement_period
DDS_DiscoveryConfigQosPolicy, 940
max_instances
DDS_DurabilityServiceQosPolicy, 1011
DDS_ResourceLimitsQosPolicy, 1311
max_interface_count
NDDS_Transport_Property_t, 2051
max_liveliness_loss_detection_period
DDS_DiscoveryConfigQosPolicy, 938
max_nack_response_delay
DDS_RtpsReliableWriterProtocol_t, 1328
max_objects_per_thread
DDS_SystemResourceLimitsQosPolicy, 1383
max_outstanding_reads
DDS_BuiltinTopicReaderResourceLimits_t, 818
DDS_DataReaderResourceLimitsQosPolicy, 873
max_partition_cumulative_characters
DDS_DomainParticipantResourceLimitsQosPolicy, 993
max_partitions
DDS_DomainParticipantResourceLimitsQosPolicy, 993
max_query_condition_filters
DDS_DataReaderResourceLimitsQosPolicy, 880
max_remote_reader_filters
DDS_DataWriterResourceLimitsQosPolicy, 920
max_remote_readers
DDS_DataWriterResourceLimitsQosPolicy, 923
max_remote_virtual_writers
DDS_DataReaderResourceLimitsQosPolicy, 878
max_remote_virtual_writers_per_instance
DDS_DataReaderResourceLimitsQosPolicy, 879
max_remote_writers
DDS_DataReaderResourceLimitsQosPolicy, 870
max_remote_writers_per_instance
DDS_DataReaderResourceLimitsQosPolicy, 870
max_remote_writers_per_sample
DDS_DataReaderResourceLimitsQosPolicy, 880
max_samples
DDS_BatchQosPolicy, 811
DDS_BuiltinTopicReaderResourceLimits_t, 816
DDS_DurabilityServiceQosPolicy, 1011
DDS_ResourceLimitsQosPolicy, 1311
max_samples_per_instance
DDS_DurabilityServiceQosPolicy, 1012
DDS_ResourceLimitsQosPolicy, 1311
max_samples_per_read
DDS_BuiltinTopicReaderResourceLimits_t, 818
DDS_DataReaderResourceLimitsQosPolicy, 873
max_samples_per_remote_writer
DDS_DataReaderResourceLimitsQosPolicy, 870
max_send_window_size
DDS_RtpsReliableWriterProtocol_t, 1333
max_size_serialized
DDS_DynamicDataTypeSerializationProperty_t, 1140
max_skiplist_level
DDS_DatabaseQosPolicy, 834
max_tokens
DDS_FlowControllerTokenBucketProperty_t, 1158
max_topic_queries
DDS_DataReaderResourceLimitsQosPolicy, 882
max_total_instances
DDS_DataReaderResourceLimitsQosPolicy, 877
max_value
DDS_TypeCode, 1468
max_virtual_writers
DDS_DataWriterResourceLimitsQosPolicy, 923

max_weak_references
DDS_DatabaseQosPolicy, 834
maximum
FooSeq, 1975
member_bitfield_bits
DDS_TypeCode, 1445
member_count
DDS_DynamicDataInfo, 1134
DDS_TypeCode, 1437
member_default_value
DDS_TypeCode, 1470
member_exists
DDS_DynamicData, 1046
DDS_DynamicDataMemberInfo, 1136
member_exists_in_type
DDS_DynamicData, 1047
member_id
DDS_DynamicDataMemberInfo, 1135
DDS_TypeCode, 1455
member_kind
DDS_DynamicDataMemberInfo, 1136
member_label
DDS_TypeCode, 1441
member_label_count
DDS_TypeCode, 1440
member_max_value
DDS_TypeCode, 1471
member_min_value
DDS_TypeCode, 1470
member_name
DDS_DynamicDataMemberInfo, 1135
DDS_TypeCode, 1438
member_ordinal
DDS_TypeCode, 1442
member_type
DDS_TypeCode, 1439
member_unit
DDS_TypeCode, 1472
member_visibility
DDS_TypeCode, 1446
message_auth
DDS_ParticipantTrustSignatureAlgorithmInfo, 1240
message_id
NDDS_Config_LogMessage, 2041
message_size_max
DDS_TransportInfo_t, 1409
NDDS_Transport_Property_t, 2047
metatraffic_transport_priority
DDS_DiscoveryQosPolicy, 963
metrics
DDS_MonitoringTelemetryData, 1213
Middleware Memory Management, 297
middleware_forwarding_level
DDS_MonitoringLoggingForwardingSettings, 1204
min_app_ack_response_keep_duration
DDS_RtpsReliableReaderProtocol_t, 1318
min_heartbeat_response_delay
DDS_RtpsReliableReaderProtocol_t, 1315
min_initial_participant_announcement_period
DDS_DiscoveryConfigQosPolicy, 940
min_nack_response_delay
DDS_RtpsReliableWriterProtocol_t, 1328
min_send_window_size
DDS_RtpsReliableWriterProtocol_t, 1333
min_size_serialized
DDS_DynamicDataTypeSerializationProperty_t, 1140
min_value
DDS_TypeCode, 1467
minimum_separation
DDS_TimeBasedFilterQosPolicy, 1391
minor
DDS_ProductVersion_t, 1256
DDS_ProtocolVersion_t, 1263
NDDS_Config_LibraryVersion_t, 2039
minor_version
connext::MessagingLibraryVersion, 2006
MONITORING, 610
DDS_MONITORING_QOS_POLICY_NAME, 611
monitoring
DDS_DomainParticipantFactoryQos, 969
Monitoring Library 2.0, 719
RTI_MONITORING_EVENT_TOPIC_NAME, 720
RTI_Monitoring_initialize, 720
RTI_MONITORING_LOGGING_TOPIC_NAME, 721
RTI_MONITORING_PERIODIC_TOPIC_NAME, 720
move
Infrastructure, 195
move_construct_from_loans
connext::LoanedSamples< T >, 1999
Multi-channel DataWriters, 170
multi_channel
DDS_DataWriterQos, 916
multicast
DDS_DataReaderQos, 864
Multicast Mapping, 661
Multicast Settings, 661
multicast_enabled
NDDS_Transport_UDPv4_Property_t, 2059
NDDS_Transport_UDPv6_Property_t, 2080
multicast_locators
DDS_SubscriptionBuiltinTopicData, 1376
multicast_loopback_disabled
NDDS_Transport_UDPv4_Property_t, 2060
NDDS_Transport_UDPv6_Property_t, 2080
multicast_mapping
DDS_DomainParticipantQos, 976
multicast_receive_addresses

DDS_DiscoveryQosPolicy, 962
multicast_resend_threshold
DDS_RtpsReliableWriterProtocol_t, 1337
multicast_settings
DDS_ChannelSettings_t, 822
multicast_ttl
NDDS_Transport_UDPv4_Property_t, 2059
NDDS_Transport_UDPv6_Property_t, 2080
MULTICHANNEL, 611
DDS_MULTICHANNEL_QOS_POLICY_NAME, 611
Mutable QoS, 369
my_complex
MyFlatFinalOffset, 2016, 2017
my_complex_array
MyFlatFinalOffset, 2016, 2017
my_final
MyFlatMutableOffset, 2028, 2030
MyFlatUnionOffset, 2037, 2038
my_final_array
MyFlatMutableOffset, 2028, 2030
my_final_seq
MyFlatMutableOffset, 2028, 2030
my_mutable
MyFlatMutableOffset, 2028, 2031
MyFlatUnionOffset, 2037, 2038
my_mutable_array
MyFlatMutableOffset, 2029, 2031
my_mutable_seq
MyFlatMutableOffset, 2029, 2031
my_optional_primitive
MyFlatMutableOffset, 2027
my_primitive
MyFlatFinalOffset, 2016, 2017
MyFlatMutableOffset, 2027, 2029
MyFlatUnionOffset, 2037, 2038
my_primitive_array
MyFlatFinalOffset, 2016, 2017
MyFlatMutableOffset, 2027, 2030
my_primitive_seq
MyFlatMutableOffset, 2028, 2030
my_string
MyFlatMutableOffset, 2029, 2031
my_string_seq
MyFlatMutableOffset, 2029, 2031
MyFlatFinal
FlatData Samples, 770
MyFlatFinalOffset, 2014
ConstOffset, 2015
my_complex, 2016, 2017
my_complex_array, 2016, 2017
my_primitive, 2016, 2017
my_primitive_array, 2016, 2017
MyFlatFinalOffset, 2016
MyFlatMutable
FlatData Samples, 770
MyFlatMutableBuilder, 2018
add_my_final, 2023
add_my_final_array, 2023
add_my_optional_primitive, 2022
add_my_primitive, 2022
add_my_primitive_array, 2022
build_my_final_seq, 2023
build_my_mutable, 2024
build_my_mutable_array, 2024
build_my_mutable_seq, 2024
build_my_primitive_seq, 2023
build_my_string, 2024
build_my_string_seq, 2024
finish, 2021
finish_sample, 2021
MyFlatMutableBuilder, 2020
Offset, 2020
MyFlatMutableOffset, 2025
ConstOffset, 2026
my_final, 2028, 2030
my_final_array, 2028, 2030
my_final_seq, 2028, 2030
my_mutable, 2028, 2031
my_mutable_array, 2029, 2031
my_mutable_seq, 2029, 2031
my_optional_primitive, 2027
my_primitive, 2027, 2029
my_primitive_array, 2027, 2030
my_primitive_seq, 2028, 2030
my_string, 2029, 2031
my_string_seq, 2029, 2031
MyFlatMutableOffset, 2027
MyFlatUnion
FlatData Samples, 770
MyFlatUnionBuilder, 2032
add_my_final, 2034
add_my_primitive, 2034
build_my_mutable, 2034
finish, 2033
finish_sample, 2033
MyFlatUnionBuilder, 2033
Offset, 2033
MyFlatUnionOffset, 2035
_d, 2036
ConstOffset, 2036
my_final, 2037, 2038
my_mutable, 2037, 2038
my_primitive, 2037, 2038
MyFlatUnionOffset, 2036

nack_period
DDS_RtpsReliableReaderProtocol_t, 1316
nack_suppression_duration

DDS_RtpsReliableWriterProtocol_t, 1329
name
DDS_EntityNameQosPolicy, 1148
DDS_EnumMember, 1149
DDS_PartitionQosPolicy, 1242
DDS_Property_t, 1260
DDS_StructMember, 1365
DDS_Tag, 1384
DDS_TopicBuiltinTopicData, 1392
DDS_TypeCode, 1436
DDS_UnionMember, 1501
DDS_ValueMember, 1506
nanosec
DDS_Duration_t, 1013
DDS_Time_t, 1388
narrow
DDSContentFilteredTopic, 1552
DDSKeyedOctetsDataReader, 1750
DDSKeyedOctetsDataWriter, 1753
DDSKeyedStringDataReader, 1778
DDSKeyedStringDataWriter, 1780
DDSMultiTopic, 1799
DDSOctetsDataReader, 1804
DDSOctetsDataWriter, 1806
DDSStringDataReader, 1851
DDSStringDataWriter, 1853
DDSTopic, 1887
FooDataReader, 1919
FooDataWriter, 1946
NDDS_CONFIG_ACTIVITY_CONTEXT_ATTRIBUTE_DOMAIN_ID
Activity Context, 733
NDDS_CONFIG_ACTIVITY_CONTEXT_ATTRIBUTE_ENTITY_KIND
Activity Context, 733
NDDS_CONFIG_ACTIVITY_CONTEXT_ATTRIBUTE_ENTITY_NAME
Activity Context, 733
NDDS_CONFIG_ACTIVITY_CONTEXT_ATTRIBUTE_GUID_PREFIX
Activity Context, 732
NDDS_CONFIG_ACTIVITY_CONTEXT_ATTRIBUTE_MASK_ALL
Activity Context, 731
NDDS_CONFIG_ACTIVITY_CONTEXT_ATTRIBUTE_MASK_DEFAULT
Activity Context, 730
NDDS_CONFIG_ACTIVITY_CONTEXT_ATTRIBUTE_MASK_NONE
Activity Context, 731
NDDS_CONFIG_ACTIVITY_CONTEXT_ATTRIBUTE_TOPIC
Activity Context, 732
NDDS_CONFIG_ACTIVITY_CONTEXT_ATTRIBUTE_TYPE
Activity Context, 732
NDDS_Config_ActivityContextAttributeKind
Activity Context, 731
NDDS_Config_ActivityContextAttributeKindMask
Activity Context, 731
NDDS_Config_LibraryVersion_t, 2039
build, 2040
major, 2039
minor, 2039
release, 2039
NDDS_CONFIG_LOG_CATEGORY_ALL
Logging, 726
NDDS_CONFIG_LOG_CATEGORY_API
Logging, 726
NDDS_CONFIG_LOG_CATEGORY_COMMUNICATION
Logging, 726
NDDS_CONFIG_LOG_CATEGORY_DATABASE
Logging, 726
NDDS_CONFIG_LOG_CATEGORY_DISCOVERY
Logging, 726
NDDS_CONFIG_LOG_CATEGORY_ENTITIES
Logging, 726
NDDS_CONFIG_LOG_CATEGORY_PLATFORM
Logging, 726
NDDS_CONFIG_LOG_CATEGORY_SECURITY
Logging, 726
NDDS_CONFIG_LOG_CATEGORY_USER
Logging, 726
NDDS_CONFIG_LOG_FACILITY_MIDDLEWARE
Logging, 727
NDDS_CONFIG_LOG_FACILITY_SECURITY_EVENT
Logging, 727
NDDS_CONFIG_LOG_FACILITY_SERVICE
Logging, 727
NDDS_CONFIG_LOG_FACILITY_USER
Logging, 727
NDDS_CONFIG_LOG_LEVEL_DEBUG
Logging, 724
NDDS_CONFIG_LOG_LEVEL_ERROR
Logging, 724
NDDS_CONFIG_LOG_LEVEL_FATAL_ERROR
Logging, 724
NDDS_CONFIG_LOG_LEVEL_STATUS_LOCAL
Logging, 724
NDDS_CONFIG_LOG_LEVEL_STATUS_REMOTE
Logging, 724
NDDS_CONFIG_LOG_LEVEL_WARNING
Logging, 724
NDDS_CONFIG_LOG_PRINT_FORMAT_DEBUG
Logging, 727
NDDS_CONFIG_LOG_PRINT_FORMAT_DEFAULT
Logging, 726
NDDS_CONFIG_LOG_PRINT_FORMAT_MAXIMAL
Logging, 727
NDDS_CONFIG_LOG_PRINT_FORMAT_MINIMAL
Logging, 727
NDDS_CONFIG_LOG_PRINT_FORMAT_TIMESTAMPED
Logging, 726
NDDS_CONFIG_LOG_PRINT_FORMAT_VERBOSE
Logging, 727
NDDS_CONFIG_LOG_PRINT_FORMAT_VERBOSE_TIMESTAMPED
Logging, 727

NDDS_CONFIG_LOG_VERBOSITY_ERROR
Logging, 724
NDDS_CONFIG_LOG_VERBOSITY_SILENT
Logging, 724
NDDS_CONFIG_LOG_VERBOSITY_STATUS_ALL
Logging, 724
NDDS_CONFIG_LOG_VERBOSITY_STATUS_LOCAL
Logging, 724
NDDS_CONFIG_LOG_VERBOSITY_STATUS_REMOTE
Logging, 724
NDDS_CONFIG_LOG_VERBOSITY_WARNING
Logging, 724
NDDS_Config_LogCategory
Logging, 725
NDDS_Config_LogFacility
Logging, 727
NDDS_Config_LogLevel
Logging, 724
NDDS_Config_LogMessage, 2040
facility, 2042
is_license_message, 2041
is_security_message, 2041
level, 2041
message_id, 2041
text, 2040
timestamp, 2042
NDDS_Config_LogPrintFormat
Logging, 726
NDDS_Config_LogVerbosity
Logging, 723
NDDS_CONFIG_RPC_COMPLIANCE_MASK_DEFAULT
Compliance Configuration, 736
NDDS_CONFIG_RPC_COMPLIANCE_MASK_VENDOR
Compliance Configuration, 736
NDDS_CONFIG_RPC_USE_STANDARD_RELATED_SAMPLE_IDENTITY
Compliance Configuration, 745
NDDS_Config_RpcComplianceMask
Compliance Configuration, 737
NDDS_Config_RpcComplianceMaskBits
Compliance Configuration, 745
NDDS_CONFIG_SYSLOG_LEVEL_ALERT
Logging, 725
NDDS_CONFIG_SYSLOG_LEVEL_CRITICAL
Logging, 725
NDDS_CONFIG_SYSLOG_LEVEL_DEBUG
Logging, 725
NDDS_CONFIG_SYSLOG_LEVEL_EMERGENCY
Logging, 725
NDDS_CONFIG_SYSLOG_LEVEL_ERROR
Logging, 725
NDDS_CONFIG_SYSLOG_LEVEL_INFORMATIONAL
Logging, 725
NDDS_CONFIG_SYSLOG_LEVEL_NOTICE
Logging, 725
NDDS_CONFIG_SYSLOG_LEVEL_WARNING
Logging, 725
NDDS_CONFIG_SYSLOG_VERBOSITY_ALERT
Logging, 728
NDDS_CONFIG_SYSLOG_VERBOSITY_CRITICAL
Logging, 728
NDDS_CONFIG_SYSLOG_VERBOSITY_DEBUG
Logging, 728
NDDS_CONFIG_SYSLOG_VERBOSITY_EMERGENCY
Logging, 728
NDDS_CONFIG_SYSLOG_VERBOSITY_ERROR
Logging, 728
NDDS_CONFIG_SYSLOG_VERBOSITY_INFORMATIONAL
Logging, 728
NDDS_CONFIG_SYSLOG_VERBOSITY_NOTICE
Logging, 728
NDDS_CONFIG_SYSLOG_VERBOSITY_SILENT
Logging, 728
NDDS_CONFIG_SYSLOG_VERBOSITY_WARNING
Logging, 728
NDDS_Config_SyslogLevel
Logging, 725
NDDS_Config_SyslogVerbosity
Logging, 727
NDDS_CONFIG_XTYPES_ACCEPT_UNKNOWN_DISCRIMINATOR_BIT
Compliance Configuration, 741
NDDS_CONFIG_XTYPES_ACCEPT_UNKNOWN_ENUM_VALUE_BIT
Compliance Configuration, 740
NDDS_CONFIG_XTYPES_COMPLIANCE_MASK_DEFAULT
Compliance Configuration, 735
NDDS_CONFIG_XTYPES_COMPLIANCE_MASK_VENDOR
Compliance Configuration, 735
NDDS_CONFIG_XTYPES_DHEADER_IN_NON_PRIMITIVE_COLLECTION
Compliance Configuration, 739
NDDS_CONFIG_XTYPES_ENCAPSULATION_OPTIONS_WITH_PADDING
Compliance Configuration, 740
NDDS_CONFIG_XTYPES_ENUM_AS_PRIMITIVE_IN_COLLECTIONS_BI
Compliance Configuration, 739
NDDS_CONFIG_XTYPES_INITIALIZE_DISCRIMINATOR_TO_DEFAULT_B
Compliance Configuration, 743
NDDS_CONFIG_XTYPES_JAVA_KEYHASH_750_BIT
Compliance Configuration, 744
NDDS_CONFIG_XTYPES_MUST_UNDERSTAND_IN_KEY_AND_DISCRIM
Compliance Configuration, 745
NDDS_CONFIG_XTYPES_PARAMETER_LENGTH_WITH_PADDING_BIT
Compliance Configuration, 739
NDDS_CONFIG_XTYPES_PYTHON_INT8_730_BIT
Compliance Configuration, 744
NDDS_CONFIG_XTYPES_SELECT_DEFAULT_DISCRIMINATOR_BIT
Compliance Configuration, 741
NDDS_CONFIG_XTYPES_SENTINEL_IN_EMPTY_UNION_BIT
Compliance Configuration, 742
NDDS_Config_XTypesComplianceMask
Compliance Configuration, 736

NDDS_Config_XTypesComplianceMaskBits
Compliance Configuration, 738
NDDS_DISCOVERY_PEERS, 662
NDDS_Transport_Address_from_string
Transport Address, 451
NDDS_TRANSPORT_ADDRESS_INVALID
Transport Address, 453
NDDS_TRANSPORT_ADDRESS_INVALID_INITIALIZER
Transport Address, 449
NDDS_Transport_Address_is_ipv4
Transport Address, 452
NDDS_Transport_Address_is_multicast
Transport Address, 453
NDDS_Transport_Address_print
Transport Address, 452
NDDS_TRANSPORT_ADDRESS_STRING_BUFFER_SIZE
Transport Address, 450
NDDS_Transport_Address_t, 2042
network_ordered_value, 2043
NDDS_Transport_Address_to_string
Transport Address, 450
NDDS_Transport_Address_to_string_with_protocol_family_format
Transport Address, 451
NDDS_TRANSPORT_ALLOCATION_SETTINGS_DEFAULT
Transport Plugins Configuration, 444
NDDS_TRANSPORT_ALLOCATION_SETTINGS_INCREMENTAL_COUNT_AUTOMATIC
Transport Plugins Configuration, 444
NDDS_TRANSPORT_ALLOCATION_SETTINGS_MAX_COUNT_UNLIMITED
Transport Plugins Configuration, 443
NDDS_TRANSPORT_CLASSID_INVALID
Transport Plugins Configuration, 444
NDDS_TRANSPORT_CLASSID_RESERVED_RANGE
Transport Plugins Configuration, 446
NDDS_TRANSPORT_CLASSID_SHMEM
Transport Plugins Configuration, 445
NDDS_TRANSPORT_CLASSID_SHMEM_510
Transport Plugins Configuration, 445
NDDS_Transport_ClassId_t
Transport Plugins Configuration, 447
NDDS_TRANSPORT_CLASSID_TCPV4_LAN
Transport Plugins Configuration, 445
NDDS_TRANSPORT_CLASSID_TCPV4_WAN
Transport Plugins Configuration, 445
NDDS_TRANSPORT_CLASSID_TLSV4_LAN
Transport Plugins Configuration, 446
NDDS_TRANSPORT_CLASSID_TLSV4_WAN
Transport Plugins Configuration, 446
NDDS_TRANSPORT_CLASSID_UDPv4
Transport Plugins Configuration, 444
NDDS_TRANSPORT_CLASSID_UDPv4_WAN
Transport Plugins Configuration, 446
NDDS_TRANSPORT_CLASSID_UDPv6
Transport Plugins Configuration, 445
NDDS_TRANSPORT_CLASSID_UDPv6_510
Transport Plugins Configuration, 445
NDDS_TRANSPORT_CLASSNAME_TCPV4_WAN
Transport Plugins Configuration, 446
NDDS_Transport_create_plugin
Installing Transport Plugins, 181
NDDS_Transport_Handle_is_nil
Installing Transport Plugins, 182
NDDS_TRANSPORT_HANDLE_NIL
Installing Transport Plugins, 182
NDDS_Transport_Handle_t
Installing Transport Plugins, 181
NDDS_TRANSPORT_INTERFACE_OFF
Interface, 441
NDDS_TRANSPORT_INTERFACE_ON
Interface, 441
NDDS_TRANSPORT_INTERFACE_RANK_UNKNOWN
Transport Plugins Configuration, 443
NDDS_Transport_Interface_Status_t
Interface, 440
NDDS_Transport_Interface_t, 2043
address, 2043
rank, 2044
status, 2044
transport_classid, 2043
NDDS_TRANSPORT_LENGTH_UNLIMITED
Transport Plugins Configuration, 443
NDDS_TRANSPORT_PORT_INVALID
Transport Plugins Configuration, 443
NDDS_Transport_Port_t
Transport Plugins Configuration, 447
NDDS_TRANSPORT_PROPERTY_BIT_BUFFER_ALWAYS_LOANED
Transport Plugins Configuration, 446
NDDS_TRANSPORT_PROPERTY_GATHER_SEND_BUFFER_COUNT_M
Transport Plugins Configuration, 447
NDDS_Transport_Property_t, 2044
address_bit_count, 2046
allow_interfaces_list, 2047
allow_interfaces_list_length, 2048
allow_multicast_interfaces_list, 2049
allow_multicast_interfaces_list_length, 2050
classid, 2045
deny_interfaces_list, 2048
deny_interfaces_list_length, 2049
deny_multicast_interfaces_list, 2050
deny_multicast_interfaces_list_length, 2050
gather_send_buffer_count_max, 2047
max_interface_count, 2051
message_size_max, 2047
properties_bitmap, 2046
thread_name_prefix, 2051
transport_uuid, 2051
NDDS_TRANSPORT_SHMEM_ADDRESS_BIT_COUNT
Shared Memory Transport, 460
NDDS_Transport_Shmem_create

Shared Memory Transport, 462
NDDS_TRANSPORT_SHMEM_GATHER_SEND_BUFFER_COUNT_MAX_DEFAULT
Shared Memory Transport, 460
NDDS_TRANSPORT_SHMEM_MAJOR_AFTER_BUG_1424_FIX
Shared Memory Transport, 461
NDDS_TRANSPORT_SHMEM_MESSAGE_SIZE_MAX_DEFAULT
Shared Memory Transport, 460
NDDS_Transport_Shmem_new
Shared Memory Transport, 461
NDDS_TRANSPORT_SHMEM_PROPERTIES_BITMAP_DEFAULT
Shared Memory Transport, 460
NDDS_TRANSPORT_SHMEM_PROPERTY_DEFAULT
Shared Memory Transport, 461
NDDS_Transport_Shmem_Property_t, 2052
enable_udp_debugging, 2054
parent, 2053
receive_buffer_size, 2053
received_message_count_max, 2053
udp_debugging_address, 2054
udp_debugging_port, 2055
NDDS_TRANSPORT_SHMEM_RECEIVE_BUFFER_SIZE_DEFAULT
Shared Memory Transport, 461
NDDS_TRANSPORT_SHMEM_RECEIVED_MESSAGE_COUNT_MAX_DEFAULT
Shared Memory Transport, 461
NDDS_TRANSPORT_UDP_GATHER_SEND_BUFFER_COUNT_MAX_DEFAULT
UDP Transport Plugin definitions, 455
NDDS_TRANSPORT_UDP_MULTICAST_TTL_DEFAULT
UDP Transport Plugin definitions, 456
NDDS_Transport_UDP_Port
UDP Transport Plugin definitions, 456
NDDS_TRANSPORT_UDP_PROPERTIES_BITMAP_DEFAULT
UDP Transport Plugin definitions, 454
NDDS_TRANSPORT_UDP_RECV_SOCKET_BUFFER_SIZE_DEFAULT
UDP Transport Plugin definitions, 456
NDDS_TRANSPORT_UDP_SEND_SOCKET_BUFFER_SIZE_DEFAULT
UDP Transport Plugin definitions, 455
NDDS_TRANSPORT_UDP_SOCKET_BUFFER_SIZE_OS_DEFAULT
UDP Transport Plugin definitions, 455
NDDS_TRANSPORT_UDP_SOCKET_BUFFER_SIZE_OS_MAX
UDP Transport Plugin definitions, 455
NDDS_Transport_UDP_WAN_CommPortsMappingInfo, 2055
host_port, 2056
public_port, 2056
rtps_port, 2055
NDDS_TRANSPORT_UDPV4_ADDRESS_BIT_COUNT
UDPv4 Transport, 467
NDDS_TRANSPORT_UDPV4_BLOCKING_ALWAYS
UDPv4 Transport, 469
NDDS_TRANSPORT_UDPV4_BLOCKING_DEFAULT
UDPv4 Transport, 469
NDDS_TRANSPORT_UDPV4_BLOCKING_NEVER
UDPv4 Transport, 469
NDDS_Transport_UDPv4_create
UDPv4 Transport, 471
NDDS_Transport_UDPv4_create_from_properties_with_prefix
UDPv4 Transport, 472
NDDS_TRANSPORT_UDPV4_GATHER_SEND_BUFFER_COUNT_MAX_DEFAULT
UDPv4 Transport, 467
NDDS_TRANSPORT_UDPV4_MESSAGE_SIZE_MAX_DEFAULT
UDPv4 Transport, 468
NDDS_TRANSPORT_UDPV4_MULTICAST_TTL_DEFAULT
UDPv4 Transport, 469
NDDS_Transport_UDPv4_new
UDPv4 Transport, 470
NDDS_TRANSPORT_UDPV4_PAYLOAD_SIZE_MAX
UDPv4 Transport, 468
NDDS_TRANSPORT_UDPV4_PROPERTIES_BITMAP_DEFAULT
UDPv4 Transport, 467
NDDS_TRANSPORT_UDPV4_PROPERTY_DEFAULT
UDPv4 Transport, 469
NDDS_Transport_UDPv4_Property_t, 2056
disable_interface_tracking, 2065
force_interface_poll_detection, 2064
ignore_loopback_interface, 2060
ignore_nonrunning_interfaces, 2061
ignore_non_up_interfaces, 2060
interface_poll_period, 2064
join_multicast_group_timeout, 2065
multicast_enabled, 2059
multicast_loopback_disabled, 2060
multicast_ttl, 2059
no_zero_copy, 2061
parent, 2058
port_offset, 2066
protocol_overhead_max, 2064
public_address, 2065
recv_socket_buffer_size, 2058
reuse_multicast_receive_resource, 2064
send_blocking, 2062
send_ping, 2063
send_socket_buffer_size, 2058
transport_priority_mapping_high, 2063
transport_priority_mapping_low, 2063
transport_priority_mask, 2062
unicast_enabled, 2059
use_checksum, 2062
NDDS_TRANSPORT_UDPV4_RECV_SOCKET_BUFFER_SIZE_DEFAULT
UDPv4 Transport, 468
NDDS_TRANSPORT_UDPV4_SEND_SOCKET_BUFFER_SIZE_DEFAULT
UDPv4 Transport, 468
NDDS_TRANSPORT_UDPV4_SOCKET_BUFFER_SIZE_OS_DEFAULT
UDPv4 Transport, 468
NDDS_TRANSPORT_UDPV4_SOCKET_BUFFER_SIZE_OS_MAX
UDPv4 Transport, 467
NDDS_Transport_UDPv4_string_to_address_cEA
UDPv4 Transport, 469
NDDS_TRANSPORT_UDPV4_WAN_ADDRESS_BIT_COUNT

UDPv4 Transport, 467
NDDS_Transport_UDPv4_WAN_create
Real-Time WAN Transport, 477
NDDS_Transport_UDPv4_WAN_create_from_properties_with_prefix
Real-Time WAN Transport, 478
NDDS_TRANSPORT_UDPV4_WAN_MESSAGE_SIZE_MAX_DEFAULT
Real-Time WAN Transport, 476
NDDS_Transport_UDPv4_WAN_new
Real-Time WAN Transport, 477
NDDS_TRANSPORT_UDPV4_WAN_PROPERTY_DEFAULT
Real-Time WAN Transport, 476
NDDS_Transport_UDPv4_WAN_Property_t, 2067
binding_ping_period, 2076
comm_ports_list, 2074
comm_ports_list_length, 2075
disable_interface_tracking, 2073
force_interface_poll_detection, 2073
ignore_loopback_interface, 2069
ignore_nonrunning_interfaces, 2070
ignore_nonup_interfaces, 2069
interface_poll_period, 2073
no_zero_copy, 2070
parent, 2068
port_offset, 2076
protocol_overhead_max, 2073
public_address, 2074
recv_socket_buffer_size, 2068
send_blocking, 2071
send_ping, 2072
send_socket_buffer_size, 2068
transport_priority_mapping_high, 2072
transport_priority_mapping_low, 2072
transport_priority_mask, 2071
use_checksum, 2071
NDDS_TRANSPORT_UDPV6_ADDRESS_BIT_COUNT
UDPv6 Transport, 483
NDDS_TRANSPORT_UDPV6_BLOCKING_ALWAYS
UDPv6 Transport, 485
NDDS_TRANSPORT_UDPV6_BLOCKING_NEVER
UDPv6 Transport, 485
NDDS_Transport_UDPv6_create
UDPv6 Transport, 487
NDDS_Transport_UDPv6_create_from_properties_with_prefix
UDPv6 Transport, 488
NDDS_TRANSPORT_UDPV6_GATHER_SEND_BUFFER_COUNT_MAX_DEFAULT
UDPv6 Transport, 483
NDDS_TRANSPORT_UDPV6_MESSAGE_SIZE_MAX_DEFAULT
UDPv6 Transport, 484
NDDS_TRANSPORT_UDPV6_MULTICAST_TTL_DEFAULT
UDPv6 Transport, 485
NDDS_Transport_UDPv6_new
UDPv6 Transport, 486
NDDS_TRANSPORT_UDPV6_PAYLOAD_SIZE_MAX
UDPv6 Transport, 484
NDDS_TRANSPORT_UDPV6_PROPERTIES_BITMAP_DEFAULT
UDPv6 Transport, 483
NDDS_Transport_UDPv6_Property_t, 2077
disable_interface_tracking, 2085
enable_v4mapped, 2082
force_interface_poll_detection, 2084
ignore_loopback_interface, 2080
ignore_nonrunning_interfaces, 2081
interface_poll_period, 2084
join_multicast_group_timeout, 2085
multicast_enabled, 2080
multicast_loopback_disabled, 2080
multicast_ttl, 2080
no_zero_copy, 2081
parent, 2078
port_offset, 2086
protocol_overhead_max, 2084
public_address, 2085
recv_socket_buffer_size, 2079
reuse_multicast_receive_resource, 2084
send_blocking, 2082
send_ping, 2083
send_socket_buffer_size, 2078
transport_priority_mapping_high, 2083
transport_priority_mapping_low, 2083
transport_priority_mask, 2082
unicast_enabled, 2079
NDDS_TRANSPORT_UDPV6_RECV_SOCKET_BUFFER_SIZE_DEFAULT
UDPv6 Transport, 484
NDDS_TRANSPORT_UDPV6_SEND_SOCKET_BUFFER_SIZE_DEFAULT
UDPv6 Transport, 484
NDDS_TRANSPORT_UDPV6_SOCKET_BUFFER_SIZE_OS_DEFAULT
UDPv6 Transport, 484
NDDS_TRANSPORT_UDPV6_SOCKET_BUFFER_SIZE_OS_MAX
UDPv6 Transport, 483
NDDS_Transport_UDPv6_string_to_address_cEA
UDPv6 Transport, 485
NDDS_Transport_UUID, 2087
NDDS_TRANSPORT_UUID_SIZE
Transport Plugins Configuration, 443
NDDS_TRANSPORT_UUID_UNKNOWN
Transport Plugins Configuration, 443
NDDS_UTILITY_HEAP_MONITORING_SNAPSHOT_CONTENT_BIT_ACTI
Heap Monitoring, 747
NDDS_UTILITY_HEAP_MONITORING_SNAPSHOT_CONTENT_BIT_FUN
Heap Monitoring, 747
NDDS_UTILITY_HEAP_MONITORING_SNAPSHOT_CONTENT_BIT_TOP
Heap Monitoring, 747
NDDS_UTILITY_HEAP_MONITORING_SNAPSHOT_CONTENT_DEFAULT
Heap Monitoring, 747
NDDS_UTILITY_HEAP_MONITORING_SNAPSHOT_CONTENT_MINIMAL
Heap Monitoring, 747
NDDS_Utility_HeapMonitoringSnapshotContentFormat
Heap Monitoring, 746

NDDS_UTILITY_NETWORK_CAPTURE_CONTENT_ENCRYPTED_DATA
Network Capture, 752
NDDS_UTILITY_NETWORK_CAPTURE_CONTENT_MASK_ALL
Network Capture, 750
NDDS_UTILITY_NETWORK_CAPTURE_CONTENT_MASK_DEFAULT
Network Capture, 750
NDDS_UTILITY_NETWORK_CAPTURE_CONTENT_MASK_NONE
Network Capture, 750
NDDS_UTILITY_NETWORK_CAPTURE_CONTENT_USER_SERIALIZED_DATA
Network Capture, 752
NDDS_UTILITY_NETWORK_CAPTURE_PARAMETERS_DEFAULT
Network Capture, 753
NDDS_UTILITY_NETWORK_CAPTURE_TRAFFIC_IN
Network Capture, 752
NDDS_UTILITY_NETWORK_CAPTURE_TRAFFIC_MASK_ALL
Network Capture, 751
NDDS_UTILITY_NETWORK_CAPTURE_TRAFFIC_MASK_DEFAULT
Network Capture, 750
NDDS_UTILITY_NETWORK_CAPTURE_TRAFFIC_MASK_NONE
Network Capture, 750
NDDS_UTILITY_NETWORK_CAPTURE_TRAFFIC_OUT
Network Capture, 752
NDDS_Utility_NetworkCaptureContentKind
Network Capture, 752
NDDS_Utility_NetworkCaptureContentKindMask
Network Capture, 751
NDDS_Utility_NetworkCaptureParams_t, 2087
checkpoint_thread_settings, 2088
dropped_content, 2088
frame_queue_size, 2089
Network Capture, 751
parse_encrypted_content, 2088
traffic, 2088
transports, 2087
NDDS_Utility_NetworkCaptureTrafficKind
Network Capture, 752
NDDS_Utility_NetworkCaptureTrafficKindMask
Network Capture, 751
NDDSConfigActivityContext, 2089
NDDSConfigCompliance, 2090
get_rpc_mask, 2091
get_xtypes_mask, 2090
load_compliance_masks, 2091
set_rpc_mask, 2092
set_xtypes_mask, 2090
NDDSConfigLogger, 2093
alert, 2098
critical, 2099
debug, 2100
emergency, 2098
error, 2099
finalize_instance, 2094
get_instance, 2094
get_output_device, 2096
get_output_file, 2096
get_print_format, 2097
get_print_format_by_log_level, 2098
get_verbosity, 2095
get_verbosity_by_category, 2095
informational, 2099
notice, 2099
set_output_device, 2097
set_output_file, 2096
set_output_file_set, 2096
set_print_format, 2097
set_print_format_by_log_level, 2098
set_verbosity, 2095
set_verbosity_by_category, 2095
warning, 2099
NDDSConfigLoggerDevice, 2100
close, 2101
write, 2101
NDDSConfigVersion, 2101
get_c_api_version, 2102
get_core_version, 2102
get_cpp_api_version, 2102
get_instance, 2102
get_product_version, 2102
to_string, 2103
NDDSLM, 2103
unload_activation_library, 2103
NDDSTransportSupport, 2104
add_receive_route, 2107
add_send_route, 2106
get_builtin_transport_property, 2108
get_transport_plugin, 2110
lookup_transport, 2106
register_transport, 2104
set_builtin_transport_property, 2109
NDDSUtility, 2110
disable_heap_monitoring, 2112
enable_heap_monitoring, 2112
get_spin_per_microsecond, 2112
pause_heap_monitoring, 2112
resume_heap_monitoring, 2113
sleep, 2111
spin, 2111
take_heap_snapshot, 2113
NDDSUtilityHeapMonitoring, 2113
disable, 2114
enable, 2114
pause, 2115
resume, 2115
take_heap_snapshot, 2115
NDDSUtilityNetworkCapture, 2117
disable, 2119
enable, 2118
pause, 2124

resume, 2125
set_default_params, 2119
start, 2120–2122
stop, 2123
Network Capture, 747
NDDS_UTILITY_NETWORK_CAPTURE_CONTENT_ENCRYPTED_DATA, 752
NDDS_UTILITY_NETWORK_CAPTURE_CONTENT_MASK_ALL, 750
NDDS_UTILITY_NETWORK_CAPTURE_CONTENT_MASK_DEFAULT, 750
NDDS_UTILITY_NETWORK_CAPTURE_CONTENT_MASK_NONE, 750
NDDS_UTILITY_NETWORK_CAPTURE_CONTENT_USER_SERIALIZED_DATA, 752
NDDS_UTILITY_NETWORK_CAPTURE_PARAMETERS_DEFAULT, 753
NDDS_UTILITY_NETWORK_CAPTURE_TRAFFIC_IN, 752
NDDS_UTILITY_NETWORK_CAPTURE_TRAFFIC_MASK_ALL, 751
NDDS_UTILITY_NETWORK_CAPTURE_TRAFFIC_MASK_DEFAULT, 750
NDDS_UTILITY_NETWORK_CAPTURE_TRAFFIC_MASK_NONE, 750
NDDS_UTILITY_NETWORK_CAPTURE_TRAFFIC_OUT, 752
NDDS_Utility_NetworkCaptureContentKind, 752
NDDS_Utility_NetworkCaptureContentKindMask, 751
NDDS_Utility_NetworkCaptureParams_t, 751
NDDS_Utility_NetworkCaptureTrafficKind, 752
NDDS_Utility_NetworkCaptureTrafficKindMask, 751
network_ordered_value
NDDS_Transport_Address_t, 2043
new_remote_participant_announcements
DDS_DiscoveryConfigQosPolicy, 939
no_writers_generation_count
DDS_SampleInfo, 1349
no_writers_instance_count
DDS_DataReaderCacheStatus, 841
no_writers_instance_count_peak
DDS_DataReaderCacheStatus, 841
no_writers_instance_removal
DDS_DataReaderResourceLimitsInstanceReplacementSettings, 867
no_zero_copy
NDDS_Transport_UDPv4_Property_t, 2061
NDDS_Transport_UDPv4_WAN_Property_t, 2070
NDDS_Transport_UDPv6_Property_t, 2081
not_alive_count
DDS_LivelinessChangedStatus, 1178
not_alive_count_change
DDS_LivelinessChangedStatus, 1178
notice
NDDSConfigLogger, 2099
notify_datareaders
DDSSubscriber, 1876

Observability Framework, 190
Octet Buffer Support, 753
DDS_OctetBuffer_alloc, 755
DDS_OctetBuffer_dup, 755
DDS_OctetBuffer_free, 756
Octets Built-in Type, 510
DDS_Octets, 510
Offset
MyFlatMutableBuilder, 2020
MyFlatUnionBuilder, 2033
rti::flat::MutableArrayBuilder< ElementBuilder, N >, 2009
rti::flat::MutableSequenceBuilder< ElementBuilder >, 2013
rti::flat::Sample< OffsetType >, 2188
Offset.hpp, 2305
old_source_timestamp_dropped_sample_count
DDS_DataReaderCacheStatus, 837
on_application_acknowledgment
DDSDataWriterListener, 1617
on_condition_triggered
DDSConditionHandler, 1547
DDSDataReaderStatusConditionHandler, 1588
on_data_available
DDSDataReaderListener, 1585
on_data_on_readers
DDSSubscriberListener, 1882
on_inconsistent_topic
DDSTopicListener, 1895
on_instance_replaced
DDSDataWriterListener, 1617
on_invalid_local_status_advance_notice
DDSDomainParticipantListener, 1724
on_liveliness_changed
DDSDataReaderListener, 1584
on_liveliness_lost
DDSDataWriterListener, 1614
on_offered_deadline_missed
DDSDataWriterListener, 1613
on_offered_incompatible_qos
DDSDataWriterListener, 1614
on_publication_matched
DDSDataWriterListener, 1615
on_reliable_reader_activity_changed
DDSDataWriterListener, 1616
on_reliable_writer_cache_changed
DDSDataWriterListener, 1615
on_request_available
connext::ReplierListener< TReq, TRep >, 2152

connext::SimpleReplierListener< TReq, TRep >, 2208
on_requested_deadline_missed
DDSDataReaderListener, 1584
on_requested_incompatible_qos
DDSDataReaderListener, 1584
on_sample_lost
DDSDataReaderListener, 1585
on_sample_rejected
DDSDataReaderListener, 1584
on_sample_removed
DDSDataWriterListener, 1616
on_service_request_accepted
DDSDataWriterListener, 1618
on_subscription_matched
DDSDataReaderListener, 1585
on_thread_deleted
DDSAsyncWaitSetListener, 1543
on_thread_spawned
DDSAsyncWaitSetListener, 1543
on_wait_timeout
DDSAsyncWaitSetListener, 1544
operator LoanMemento
connext::LoanedSamples< T >, 2002
operator T&
connext::SampleRef< T >, 2195
operator!=
DDS_DataReaderQos, 861
DDS_DataWriterQos, 911
DDS_DomainParticipantFactoryQos, 968
DDS_DomainParticipantQos, 973
DDS_PublisherQos, 1277
DDS_SubscriberQos, 1368
DDS_TopicQos, 1399
rti::flat::OffsetBase, 2131
rti::flat::SequenceIterator< E, OffsetKind >, 2204
operator<
rti::flat::OffsetBase, 2129
rti::flat::SequenceIterator< E, OffsetKind >, 2203
operator<=
rti::flat::OffsetBase, 2130
rti::flat::SequenceIterator< E, OffsetKind >, 2203
operator>
rti::flat::OffsetBase, 2129
rti::flat::SequenceIterator< E, OffsetKind >, 2203
operator>=
rti::flat::OffsetBase, 2130
rti::flat::SequenceIterator< E, OffsetKind >, 2203
operator*
rti::flat::SequenceIterator< E, OffsetKind >, 2201
operator()
connext::IsReplyRelatedPredicate< T >, 1994
operator++
connext::SampleIterator< T, IsConst >, 2191
rti::flat::SequenceIterator< E, OffsetKind >, 2202
operator->
rti::flat::SequenceIterator< E, OffsetKind >, 2202
operator=
connext::SampleRef< T >, 2194
DDS_DynamicData, 1032
FooSeq, 1968
operator==
DDS_DataReaderQos, 860
DDS_DataWriterQos, 911
DDS_DomainParticipantFactoryQos, 968
DDS_DomainParticipantQos, 973
DDS_DynamicData, 1032
DDS_PublisherQos, 1277
DDS_SubscriberQos, 1368
DDS_TopicQos, 1399
rti::flat::OffsetBase, 2130
rti::flat::SequenceIterator< E, OffsetKind >, 2204
operator[]
connext::LoanedSamples< T >, 2001
FooSeq, 1972
ordered_access
DDS_PresentationQosPolicy, 1252
ordinal
DDS_EnumMember, 1150
original_publication_virtual_guid
DDS_SampleInfo, 1352
original_publication_virtual_sequence_number
DDS_SampleInfo, 1352
original_related_reader_guid
DDS_TopicQueryData, 1403
Other QoS Attributes, 375
Other Utilities, 753
out_of_range_rejected_sample_count
DDS_DataReaderProtocolStatus, 856
output_file
DDS_LoggingQosPolicy, 1189
output_file_suffix
DDS_LoggingQosPolicy, 1190
outstanding_asynchronous_sample_allocation
DDS_DomainParticipantResourceLimitsQosPolicy, 985
OWNERSHIP, 612
DDS_EXCLUSIVE_OWNERSHIP_QOS, 613
DDS_OWNERSHIP_QOS_POLICY_NAME, 613
DDS_OwnershipQosPolicyKind, 612
DDS_SHARED_OWNERSHIP_QOS, 613
ownership
DDS_DataReaderQos, 863
DDS_DataWriterQos, 914
DDS_PublicationBuiltinTopicData, 1268
DDS_SubscriptionBuiltinTopicData, 1374
DDS_TopicBuiltinTopicData, 1395
DDS_TopicQos, 1402

ownership_dropped_sample_count
DDS_DataReaderCacheStatus, 838
OWNERSHIP_STRENGTH, 613
DDS_OWNERSHIPSTRENGTH_QOS_POLICY_NAME, 614
ownership_strength
DDS_DataWriterQos, 914
DDS_PublicationBuiltinTopicData, 1268

parent
NDDS_Transport_Shmem_Property_t, 2053
NDDS_Transport_UDPv4_Property_t, 2058
NDDS_Transport_UDPv4_WAN_Property_t, 2068
NDDS_Transport_UDPv6_Property_t, 2078
parse_encrypted_content
NDDS_Utility_NetworkCaptureParams_t, 2088
partial_configuration
DDS_ParticipantBuiltinTopicData, 1234
Participant Built-in Topics, 490
DDS_PARTICIPANT_TOPIC_NAME, 491
DDS_ParticipantBuiltinTopicData, 491
Participant Use Cases, 394
participant_announcement_period
DDS_DiscoveryConfigQosPolicy, 937
participant_configuration_reader
DDS_DiscoveryConfigQosPolicy, 959
participant_configuration_reader_resource_limits
DDS_DiscoveryConfigQosPolicy, 959
participant_configuration_writer
DDS_DiscoveryConfigQosPolicy, 957
participant_configuration_writer_data_lifecycle
DDS_DiscoveryConfigQosPolicy, 958
participant_configuration_writer_publish_mode
DDS_DiscoveryConfigQosPolicy, 957
participant_id
DDS_WireProtocolQosPolicy, 1514
participant_id_gain
DDS_RtpsWellKnownPorts_t, 1342
participant_key
DDS_PublicationBuiltinTopicData, 1266
DDS_SubscriptionBuiltinTopicData, 1372
participant_liveliness_assert_period
DDS_DiscoveryConfigQosPolicy, 937
participant_liveliness_lease_duration
DDS_DiscoveryConfigQosPolicy, 937
participant_message_reader
DDS_DiscoveryConfigQosPolicy, 947
participant_message_reader_reliability_kind
DDS_DiscoveryConfigQosPolicy, 946
participant_message_writer
DDS_DiscoveryConfigQosPolicy, 947
participant_name
DDS_DomainParticipantConfigParams_t, 965
DDS_DomainParticipantQos, 976
DDS_ParticipantBuiltinTopicData, 1232
participant_property_list_max_length
DDS_DomainParticipantResourceLimitsQosPolicy, 998
participant_property_string_max_length
DDS_DomainParticipantResourceLimitsQosPolicy, 998
participant_qos_library_name
DDS_DomainParticipantConfigParams_t, 965
participant_qos_profile_name
DDS_DomainParticipantConfigParams_t, 966
DDS_MonitoringDedicatedParticipantSettings, 1194
participant_reader_resource_limits
DDS_DiscoveryConfigQosPolicy, 941
participant_user_data_max_length
DDS_DomainParticipantResourceLimitsQosPolicy, 991
PARTITION, 614
DDS_PARTITION_QOS_POLICY_NAME, 614
partition
DDS_DomainParticipantQos, 976
DDS_ParticipantBuiltinTopicData, 1233
DDS_PublicationBuiltinTopicData, 1269
DDS_PublisherQos, 1278
DDS_SubscriberQos, 1369
DDS_SubscriptionBuiltinTopicData, 1375
pause
NDDSUtilityHeapMonitoring, 2115
NDDSUtilityNetworkCapture, 2124
pause_heap_monitoring
NDDSUtility, 2112
period
DDS_DeadlineQosPolicy, 930
DDS_FlowControllerTokenBucketProperty_t, 1159
periodic_settings
DDS_MonitoringDistributionSettings, 1197
plain_cast
FlatData Offsets, 773, 775
plugin_bitmask
DDS_EndpointTrustProtectionInfo, 1145
DDS_ParticipantTrustProtectionInfo, 1239
plugin_data
DDS_TypeSupportQosPolicy, 1500
pointer
rti::flat::SequenceIterator< E, OffsetKind >, 2200
policies
DDS_OfferedIncompatibleQosStatus, 1222
DDS_RequestedIncompatibleQosStatus, 1308
policy_id
DDS_QosPolicyCount, 1284
polling_period
DDS_MonitoringPeriodicDistributionSettings, 1210
port
DDS_Locator_t, 1184

port_base
DDS_RtpsWellKnownPorts_t, 1340
port_offset
NDDS_Transport_UDPv4_Property_t, 2066
NDDS_Transport_UDPv4_WAN_Property_t, 2076
NDDS_Transport_UDPv6_Property_t, 2086
PRESENTATION, 615
DDS_GROUP_PRESENTATION_QOS, 616
DDS_HIGHEST_OFFERED_PRESENTATION_QOS, 616
DDS_INSTANCE_PRESENTATION_QOS, 616
DDS_PRESENTATION_QOS_POLICY_NAME, 616
DDS_PresentationQosPolicyAccessScopeKind, 615
DDS_TOPIC_PRESENTATION_QOS, 616
presentation
DDS_PublicationBuiltinTopicData, 1268
DDS_PublisherQos, 1278
DDS_SubscriberQos, 1369
DDS_SubscriptionBuiltinTopicData, 1374
pretty_print
DDS_PrintFormatProperty, 1254
prevent_type_widening
DDS_TypeConsistencyEnforcementQosPolicy, 1496
print
Data Writers, 117
DataReaders, 146
DDS_DynamicData, 1040
DDS_TypeCode, 1474
DomainParticipantFactory, 45
DomainParticipants, 53
Publishers, 109
Subscribers, 132
Topics, 67
print_complete_type
DDS_TypeCodePrintFormatProperty, 1492
print_data
DDSDynamicDataTypeSupport, 1729
DDSKeyedOctetsTypeSupport, 1767
DDSKeyedStringTypeSupport, 1792
DDSOctetsTypeSupport, 1814
DDSStringTypeSupport, 1858
FooTypeSupport, 1988
print_format
DDS_LoggingQosPolicy, 1189
print_IDL
DDS_TypeCode, 1474
print_kind
DDS_TypeCodePrintFormatProperty, 1492
print_ordinals
DDS_TypeCodePrintFormatProperty, 1491
print_private
DDS_QosPrintFormat, 1286
priority
DDS_ChannelSettings_t, 823
DDS_PublishModeQosPolicy, 1282
DDS_ThreadSettings_t, 1386
DDS_WriteParams_t, 1521
product_version
DDS_ParticipantBuiltinTopicData, 1232
DDS_PublicationBuiltinTopicData, 1271
DDS_SubscriptionBuiltinTopicData, 1378
PROFILE, 616
DDS_PROFILE_QOS_POLICY_NAME, 617
profile
DDS_DomainParticipantFactoryQos, 969
Programming How-To's, 439
propagate
DDS_Property_t, 1260
propagate_app_ack_with_no_response
DDS_DataWriterProtocolQosPolicy, 896
propagate_dispose_of_unregistered_instances
DDS_DataReaderProtocolQosPolicy, 845
propagate_unregister_of_disposed_instances
DDS_DataReaderProtocolQosPolicy, 846
properties_bitmap
NDDS_Transport_Property_t, 2046
PROPERTY, 617
add_pointer_property, 624
add_property, 622
assert_pointer_property, 623
assert_property, 622
DDS_PROPERTY_QOS_IMMUTABLE, 620
DDS_PROPERTY_QOS_MUTABLE, 620
DDS_PROPERTY_QOS_MUTABLE_UNTIL_ENABLE, 620
DDS_PROPERTY_QOS_POLICY_NAME, 626
DDS_PropertyQosPolicy, 619
DDS_PropertyQosPolicyHelper_get_property_mutability, 621
DDS_PropertyQosPolicyMutability, 620
get_number_of_properties, 621
get_properties, 626
lookup_property, 625
remove_property, 625
property
DDS_DataReaderQos, 865
DDS_DataWriterQos, 916
DDS_DomainParticipantQos, 975
DDS_ParticipantBuiltinTopicData, 1230
DDS_PublicationBuiltinTopicData, 1270
DDS_SubscriptionBuiltinTopicData, 1376
protocol
DDS_DataReaderQos, 864
DDS_DataWriterQos, 915
protocol_overhead_max
NDDS_Transport_UDPv4_Property_t, 2064
NDDS_Transport_UDPv4_WAN_Property_t, 2073
NDDS_Transport_UDPv6_Property_t, 2084

public_address
NDDS_Transport_UDPv4_Property_t, 2065
NDDS_Transport_UDPv4_WAN_Property_t, 2074
NDDS_Transport_UDPv6_Property_t, 2085
public_port
NDDS_Transport_UDP_WAN_CommPortsMappingInfo, 2056
Publication Built-in Topics, 493
DDS_PUBLICATION_TOPIC_NAME, 494
DDS_PublicationBuiltinTopicData, 494
Publication Example, 392
Publication Module, 107
publication_handle
DDS_SampleInfo, 1349
publication_name
DDS_DataWriterQos, 916
DDS_PublicationBuiltinTopicData, 1272
publication_period
DDS_MonitoringEventDistributionSettings, 1200
DDS_MonitoringLoggingDistributionSettings, 1203
DDS_TopicQueryDispatchQosPolicy, 1405
publication_reader
DDS_DiscoveryConfigQosPolicy, 941
publication_reader_resource_limits
DDS_DiscoveryConfigQosPolicy, 942
publication_sequence_number
DDS_SampleInfo, 1351
publication_writer
DDS_DiscoveryConfigQosPolicy, 943
publication_writer_data_lifecycle
DDS_DiscoveryConfigQosPolicy, 944
publication_writer_publish_mode
DDS_DiscoveryConfigQosPolicy, 948
PUBLISH_MODE, 627
DDS_ASYNCHRONOUS_PUBLISH_MODE_QOS, 629
DDS_PUBLICATION_PRIORITY_AUTOMATIC, 628
DDS_PUBLICATION_PRIORITY_UNDEFINED, 628
DDS_PUBLISHMODE_QOS_POLICY_NAME, 629
DDS_PublishModeQosPolicyKind, 628
DDS_SYNCHRONOUS_PUBLISH_MODE_QOS, 629
publish_mode
DDS_DataWriterQos, 915
publisher
connext::ReplierParams< TReq, TRep >, 2155
connext::RequesterParams, 2181
connext::SimpleReplierParams< TReq, TRep >, 2212
Publisher Use Cases, 399
publisher_group_data_max_length
DDS_DomainParticipantResourceLimitsQosPolicy, 992
publisher_key
DDS_PublicationBuiltinTopicData, 1270
publisher_name
DDS_PublisherQos, 1279
publisher_qos_profile_name
DDS_MonitoringDistributionSettings, 1196
Publishers, 107
DDS_DATAWRITER_QOS_DEFAULT, 114
DDS_DATAWRITER_QOS_USE_TOPIC_QOS, 114
DDS_PublisherQos_equals, 108
print, 109
to_string, 109–113
pulled_fragment_bytes
DDS_DataWriterProtocolStatus, 908
pulled_fragment_count
DDS_DataWriterProtocolStatus, 908
pulled_sample_bytes
DDS_DataWriterProtocolStatus, 903
pulled_sample_bytes_change
DDS_DataWriterProtocolStatus, 903
pulled_sample_count
DDS_DataWriterProtocolStatus, 902
pulled_sample_count_change
DDS_DataWriterProtocolStatus, 902
push_on_write
DDS_DataWriterProtocolQosPolicy, 894
pushed_fragment_bytes
DDS_DataWriterProtocolStatus, 907
pushed_fragment_count
DDS_DataWriterProtocolStatus, 907
pushed_sample_bytes
DDS_DataWriterProtocolStatus, 900
pushed_sample_bytes_change
DDS_DataWriterProtocolStatus, 900
pushed_sample_count
DDS_DataWriterProtocolStatus, 900
pushed_sample_count_change
DDS_DataWriterProtocolStatus, 900

QoS, 549
QoS by Category, 560
QoS Mutability, 333
QoS Profile Management, 312
qos_profile
connext::ReplierParams< TReq, TRep >, 2154
connext::RequesterParams, 2180
connext::SimpleReplierParams< TReq, TRep >, 2211
Queries and Filters Syntax, 184
Query Conditions, 152
query_condition_allocation
DDS_DomainParticipantResourceLimitsQosPolicy, 985
query_expression
DDS_QueryConditionParams, 1288

query_parameters
DDS_QueryConditionParams, 1288
quorum_count
DDS_EndpointGroup_t, 1142

rank
NDDS_Transport_Interface_t, 2044
reachability_lease_duration
DDS_ParticipantBuiltinTopicData, 1233
read
DDSKeyedOctetsDataReader, 1744
DDSKeyedStringDataReader, 1772
DDSOctetsDataReader, 1802
DDSStringDataReader, 1849
FooDataReader, 1919
Read Conditions, 151
read_condition_allocation
DDS_DomainParticipantResourceLimitsQosPolicy, 985
read_instance
DDSKeyedOctetsDataReader, 1745
DDSKeyedStringDataReader, 1773
FooDataReader, 1929
read_instance_w_condition
DDSKeyedOctetsDataReader, 1746
DDSKeyedStringDataReader, 1774
FooDataReader, 1932
read_next_instance
DDSKeyedOctetsDataReader, 1747
DDSKeyedStringDataReader, 1775
FooDataReader, 1935
read_next_instance_w_condition
DDSKeyedOctetsDataReader, 1747
DDSKeyedStringDataReader, 1775
FooDataReader, 1938
read_next_sample
DDSKeyedOctetsDataReader, 1745
DDSKeyedStringDataReader, 1773
DDSOctetsDataReader, 1803
DDSStringDataReader, 1850
FooDataReader, 1927
read_replies
connext::Requester< TReq, TRep >, 2175, 2176
read_reply
connext::Requester< TReq, TRep >, 2174–2176
read_request
connext::Replier< TReq, TRep >, 2150
read_requests
connext::Replier< TReq, TRep >, 2150
read_w_condition
DDSKeyedOctetsDataReader, 1744
DDSKeyedStringDataReader, 1772
DDSOctetsDataReader, 1802
DDSStringDataReader, 1849
FooDataReader, 1925
reader_checkpoint_frequency
DDS_PersistentStorageSettings, 1248
READER_DATA_LIFECYCLE, 630
DDS_READERDATALIFECYCLE_QOS_POLICY_NAME, 630
reader_data_lifecycle
DDS_DataReaderQos, 863
reader_data_tag_list_max_length
DDS_DomainParticipantResourceLimitsQosPolicy, 1003
reader_data_tag_string_max_length
DDS_DomainParticipantResourceLimitsQosPolicy, 1003
reader_property_list_max_length
DDS_DomainParticipantResourceLimitsQosPolicy, 999
reader_property_string_max_length
DDS_DomainParticipantResourceLimitsQosPolicy, 1000
reader_resource_limits
DDS_DataReaderQos, 864
reader_user_data_max_length
DDS_DomainParticipantResourceLimitsQosPolicy, 993
Real-Time WAN Transport, 472
NDDS_Transport_UDPv4_WAN_create, 477
NDDS_Transport_UDPv4_WAN_create_from_properties_with_prefix, 478
NDDS_TRANSPORT_UDPV4_WAN_MESSAGE_SIZE_MAX_DEFAUL 476
NDDS_Transport_UDPv4_WAN_new, 477
NDDS_TRANSPORT_UDPV4_WAN_PROPERTY_DEFAULT, 476
reassembled_sample_count
DDS_DataReaderProtocolStatus, 857
receive_address
DDS_TransportMulticastSettings_t, 1418
receive_buffer_size
NDDS_Transport_Shmem_Property_t, 2053
receive_port
DDS_TransportMulticastSettings_t, 1418
DDS_TransportUnicastSettings_t, 1425
receive_replies
connext::Requester< TReq, TRep >, 2165, 2166
receive_reply
connext::Requester< TReq, TRep >, 2164, 2165
receive_request
connext::Replier< TReq, TRep >, 2146, 2147
receive_requests
connext::Replier< TReq, TRep >, 2147
receive_window_size
DDS_RtpsReliableReaderProtocol_t, 1317
received_ack_bytes

DDS_DataWriterProtocolStatus, 903
received_ack_bytes_change
DDS_DataWriterProtocolStatus, 904
received_ack_count
DDS_DataWriterProtocolStatus, 903
received_ack_count_change
DDS_DataWriterProtocolStatus, 903
received_fragment_count
DDS_DataReaderProtocolStatus, 857
received_gap_bytes
DDS_DataReaderProtocolStatus, 855
received_gap_bytes_change
DDS_DataReaderProtocolStatus, 855
received_gap_count
DDS_DataReaderProtocolStatus, 854
received_gap_count_change
DDS_DataReaderProtocolStatus, 855
received_heartbeat_bytes
DDS_DataReaderProtocolStatus, 853
received_heartbeat_bytes_change
DDS_DataReaderProtocolStatus, 853
received_heartbeat_count
DDS_DataReaderProtocolStatus, 852
received_heartbeat_count_change
DDS_DataReaderProtocolStatus, 852
received_message_count_max
NDDS_Transport_Shmem_Property_t, 2053
received_nack_bytes
DDS_DataWriterProtocolStatus, 904
received_nack_bytes_change
DDS_DataWriterProtocolStatus, 904
received_nack_count
DDS_DataWriterProtocolStatus, 904
received_nack_count_change
DDS_DataWriterProtocolStatus, 904
received_nack_fragment_bytes
DDS_DataWriterProtocolStatus, 908
received_nack_fragment_count
DDS_DataWriterProtocolStatus, 908
received_sample_bytes
DDS_DataReaderProtocolStatus, 850
received_sample_bytes_change
DDS_DataReaderProtocolStatus, 850
received_sample_count
DDS_DataReaderProtocolStatus, 849
received_sample_count_change
DDS_DataReaderProtocolStatus, 850
RECEIVER_POOL, 630
DDS_LENGTH_AUTO, 631
DDS_RECEIVERPOOL_QOS_POLICY_NAME, 631
receiver_pool
DDS_DomainParticipantQos, 975
reception_sequence_number
DDS_SampleInfo, 1352
reception_timestamp
DDS_SampleInfo, 1351
recv_socket_buffer_size
NDDS_Transport_UDPv4_Property_t, 2058
NDDS_Transport_UDPv4_WAN_Property_t, 2068
NDDS_Transport_UDPv6_Property_t, 2079
reference
rti::flat::SequenceIterator< E, OffsetKind >, 2200
register_contentfilter
DDSDomainParticipant, 1632
register_durable_subscription
DDSDomainParticipant, 1666
register_instance
DDSKeyedOctetsDataWriter, 1753
DDSKeyedStringDataWriter, 1781
FooDataWriter, 1946
register_instance_w_params
FooDataWriter, 1948
register_instance_w_timestamp
DDSKeyedOctetsDataWriter, 1754
DDSKeyedStringDataWriter, 1781
FooDataWriter, 1947
register_transport
NDDSTransportSupport, 2104
register_type
DDSDynamicDataTypeSupport, 1727
DDSKeyedOctetsTypeSupport, 1764
DDSKeyedStringTypeSupport, 1789
DDSOctetsTypeSupport, 1812
DDSStringTypeSupport, 1856
FooTypeSupport, 1981
register_type_support
DDSDomainParticipantFactory, 1720
rejected_sample_count
DDS_DataReaderProtocolStatus, 855
DDS_DataWriterProtocolStatus, 905
rejected_sample_count_change
DDS_DataReaderProtocolStatus, 855
DDS_DataWriterProtocolStatus, 905
related_datareader_key
DDS_PublicationBuiltinTopicData, 1273
related_datawriter_key
DDS_SubscriptionBuiltinTopicData, 1379
related_identity
connext::Sample< T >, 2185
connext::SampleRef< T >, 2197
related_original_publication_virtual_guid
DDS_SampleInfo, 1352
related_original_publication_virtual_sequence_number
DDS_SampleInfo, 1353
related_reader_guid
DDS_FilterSampleInfo, 1154
DDS_WriteParams_t, 1523
related_sample_identity

DDS_FilterSampleInfo, 1154
DDS_WriteParams_t, 1520
related_source_guid
DDS_FilterSampleInfo, 1154
DDS_SampleInfo, 1353
DDS_WriteParams_t, 1523
related_subscription_guid
DDS_SampleInfo, 1354
related_topic_name
DDS_ContentFilterProperty_t, 829
release
connext::LoanedSamples< T >, 2000
DDS_ProductVersion_t, 1256
NDDS_Config_LibraryVersion_t, 2039
release_version
connext::MessagingLibraryVersion, 2006
RELIABILITY, 631
DDS_APPLICATION_AUTO_ACKNOWLEDGMENT_MODE, 633
DDS_APPLICATION_EXPLICIT_ACKNOWLEDGMENT_MODE, 634
DDS_BEST_EFFORT_RELIABILITY_QOS, 633
DDS_InstanceStateConsistencyKind, 634
DDS_NO_RECOVER_INSTANCE_STATE_CONSISTENCY, 634
DDS_PROTOCOL_ACKNOWLEDGMENT_MODE, 633
DDS_RECOVER_INSTANCE_STATE_CONSISTENCY, 634
DDS_RELIABILITY_QOS_POLICY_NAME, 635
DDS_ReliabilityQosPolicyAcknowledgmentModeKind, 633
DDS_ReliabilityQosPolicyKind, 632
DDS_RELIABLE_RELIABILITY_QOS, 633
reliability
DDS_DataReaderQos, 862
DDS_DataWriterQos, 913
DDS_PublicationBuiltinTopicData, 1267
DDS_SubscriptionBuiltinTopicData, 1373
DDS_TopicBuiltinTopicData, 1394
DDS_TopicQos, 1401
Reliability QoS, 314
reload_profiles
DDSDomainParticipantFactory, 1715
remote_participant_allocation
DDS_DomainParticipantResourceLimitsQosPolicy, 983
remote_participant_hash_buckets
DDS_DomainParticipantResourceLimitsQosPolicy, 988
remote_participant_purge_kind
DDS_DiscoveryConfigQosPolicy, 938
remote_reader_allocation
DDS_DomainParticipantResourceLimitsQosPolicy, 982
remote_reader_hash_buckets
DDS_DomainParticipantResourceLimitsQosPolicy, 988
remote_topic_query_allocation
DDS_DomainParticipantResourceLimitsQosPolicy, 1001
remote_topic_query_hash_buckets
DDS_DomainParticipantResourceLimitsQosPolicy, 1002
remote_writer_allocation
DDS_DomainParticipantResourceLimitsQosPolicy, 982
remote_writer_hash_buckets
DDS_DomainParticipantResourceLimitsQosPolicy, 988
remove_from_expression_parameter
DDSContentFilteredTopic, 1555
remove_peer
DDSDomainParticipant, 1679
remove_property
PROPERTY, 625
remove_tag
DATA_TAG, 578
replace_auto
DDS_WriteParams_t, 1520
replace_empty_instances
DDS_DataWriterResourceLimitsQosPolicy, 922
replaced_dropped_sample_count
DDS_DataReaderCacheStatus, 839
replaced_unacknowledged_sample_count
DDS_ReliableWriterCacheChangedStatus, 1304
Replier, 192
connext::Replier< TReq, TRep >, 2143, 2144
replier_listener
connext::ReplierParams< TReq, TRep >, 2153
ReplierParams
connext::ReplierParams< TReq, TRep >, 2153
Reply
connext::Replier< TReq, TRep >, 2142
connext::Requester< TReq, TRep >, 2160
reply_topic_name
connext::ReplierParams< TReq, TRep >, 2154
connext::RequesterParams, 2180
connext::SimpleReplierParams< TReq, TRep >, 2211
reply_type_support
connext::ReplierParams< TReq, TRep >, 2156
connext::RequesterParams, 2182
connext::SimpleReplierParams< TReq, TRep >, 2213
ReplyDataReader
connext::Requester< TReq, TRep >, 2161

ReplyDataWriter
connext::Replier< TReq, TRep >, 2142
ReplyTypeSupport
connext::Requester< TReq, TRep >, 2161
representation
DDS_DataReaderQos, 863
DDS_DataWriterQos, 914
DDS_PublicationBuiltinTopicData, 1269
DDS_SubscriptionBuiltinTopicData, 1375
DDS_TopicBuiltinTopicData, 1395
DDS_TopicQos, 1402
Request
connext::Replier< TReq, TRep >, 2142
connext::Requester< TReq, TRep >, 2160
Request-Reply Examples, 419
Request-Reply Pattern, 191
request_body
DDS_ServiceRequest, 1360
request_topic_name
connext::ReplierParams< TReq, TRep >, 2154
connext::RequesterParams, 2180
connext::SimpleReplierParams< TReq, TRep >, 2211
request_type_support
connext::ReplierParams< TReq, TRep >, 2156
connext::RequesterParams, 2182
connext::SimpleReplierParams< TReq, TRep >, 2213
request_types_filter
DDS_DiscoveryConfigQosPolicy, 959
RequestDataReader
connext::Replier< TReq, TRep >, 2142
RequestDataWriter
connext::Requester< TReq, TRep >, 2161
Requester, 191
connext::Requester< TReq, TRep >, 2161, 2162
RequesterParams
connext::RequesterParams, 2179
RequestTypeSupport
connext::Requester< TReq, TRep >, 2161
required_mask
DDS_EndpointTrustInterceptorAlgorithmInfo, 1144
DDS_TrustAlgorithmRequirements, 1426
required_matched_endpoint_groups
DDS_AvailabilityQosPolicy, 809
RESOURCE_LIMITS, 635
DDS_LENGTH_UNLIMITED, 636
DDS_RESOURCELIMITS_QOS_POLICY_NAME, 635
resource_limits
DDS_DataReaderQos, 862
DDS_DataWriterQos, 913
DDS_DomainParticipantFactoryQos, 969
DDS_DomainParticipantQos, 974
DDS_TopicBuiltinTopicData, 1395
DDS_TopicQos, 1401
resource_selection
DDS_MonitoringMetricSelection, 1206
response_data
DDS_AcknowledgmentInfo, 795
restore
DDS_PersistentStorageSettings, 1246
resume
NDDSUtilityHeapMonitoring, 2115
NDDSUtilityNetworkCapture, 2125
resume_endpoint_discovery
DDSDomainParticipant, 1668
resume_heap_monitoring
NDDSUtility, 2113
resume_publications
DDSPublisher, 1831
Return Codes, 531
DDS_RETCODE_ALREADY_DELETED, 533
DDS_RETCODE_BAD_PARAMETER, 532
DDS_RETCODE_ERROR, 532
DDS_RETCODE_ILLEGAL_OPERATION, 533
DDS_RETCODE_IMMUTABLE_POLICY, 533
DDS_RETCODE_INCONSISTENT_POLICY, 533
DDS_RETCODE_NO_DATA, 533
DDS_RETCODE_NOT_ALLOWED_BY_SECURITY, 533
DDS_RETCODE_NOT_ENABLED, 533
DDS_RETCODE_OK, 532
DDS_RETCODE_OUT_OF_RESOURCES, 533
DDS_RETCODE_PRECONDITION_NOT_MET, 532
DDS_RETCODE_TIMEOUT, 533
DDS_RETCODE_UNSUPPORTED, 532
DDS_ReturnCode_t, 532
return_loan
connext::LoanedSamples< T >, 2002
connext::SimpleReplierListener< TReq, TRep >, 2209
DDSKeyedOctetsDataReader, 1748
DDSKeyedStringDataReader, 1776
DDSOctetsDataReader, 1803
DDSStringDataReader, 1851
FooDataReader, 1940
reuse_multicast_receive_resource
NDDS_Transport_UDPv4_Property_t, 2064
NDDS_Transport_UDPv6_Property_t, 2084
revision
DDS_ProductVersion_t, 1256
role_name
DDS_EndpointGroup_t, 1142
DDS_EntityNameQosPolicy, 1148
root
rti::flat::Sample< OffsetType >, 2189
round_trip_time

DDS_RtpsReliableReaderProtocol_t, 1317
rti, 781
RTI Connext DDS API Reference, 434
RTI Connext Exceptions, 39
rti::flat, 781
rti::flat::AbstractAlignedList< ElementOffset >, 785
begin, 786
end, 786
iterator, 786
rti::flat::AbstractBuilder, 787
~AbstractBuilder, 787
capacity, 788
check_failure, 789
discard, 788
is_nested, 788
is_valid, 788
rti::flat::AbstractListBuilder< DHeaderGen >, 789
element_count, 790
rti::flat::AbstractPrimitiveList< T >, 790
get_element, 791
set_element, 791
rti::flat::AbstractSequenceBuilder< DHeaderGen >, 792
element_count, 792
rti::flat::AggregationBuilder, 793
rti::flat::FinalAlignedArrayOffset< ElementOffset, N >, 1909
get_element, 1911
rti::flat::FinalArrayOffset< ElementOffset, N >, 1911
get_element, 1912
rti::flat::FinalOffset< T >, 1913
rti::flat::FinalSequenceBuilder< ElementOffset >, 1913
add_n, 1915
add_next, 1914
finish, 1915
rti::flat::flat_type_traits< T >, 1915
rti::flat::MutableArrayBuilder< ElementBuilder, N >, 2008
build_next, 2009
finish, 2009
Offset, 2009
rti::flat::MutableArrayOffset< ElementOffset, N >, 2010
get_element, 2011
rti::flat::MutableOffset, 2011
rti::flat::MutableSequenceBuilder< ElementBuilder >, 2012
build_next, 2013
finish, 2013
Offset, 2013
rti::flat::OffsetBase, 2126
get_buffer, 2128
get_buffer_size, 2129
is_cpp_compatible, 2128
is_null, 2128
operator!=, 2131
operator<, 2129
operator<=, 2130
operator>, 2129
operator>=, 2130
operator==, 2130
rti::flat::PrimitiveArrayOffset< T, N >, 2132
element_count, 2133
rti::flat::PrimitiveConstOffset< T >, 2133
get, 2134
rti::flat::PrimitiveOffset< T >, 2134
set, 2135
rti::flat::PrimitiveSequenceBuilder< T >, 2135
add_n, 2136, 2137
add_next, 2136
finish, 2138
rti::flat::PrimitiveSequenceOffset< T >, 2138
element_count, 2139
rti::flat::Sample< OffsetType >, 2187
clone, 2190
ConstOffset, 2188
create_data, 2189
delete_data, 2190
Offset, 2188
root, 2189
rti::flat::SequenceIterator< E, OffsetKind >, 2198
advance, 2202
difference_type, 2201
is_null, 2201
iterator_category, 2200
operator!=, 2204
operator<, 2203
operator<=, 2203
operator>, 2203
operator>=, 2203
operator*, 2201
operator++, 2202
operator->, 2202
operator==, 2204
pointer, 2200
reference, 2200
SequenceIterator, 2201
value_type, 2200
rti::flat::SequenceOffset< ElementOffset >, 2204
element_count, 2206
get_element, 2205
rti::flat::StringBuilder, 2214
finish, 2215
set_string, 2215
rti::flat::StringOffset, 2216
element_count, 2217
get_string, 2216
rti::flat::UnionBuilder< Discriminator >, 2218
RTI_MONITORING_EVENT_TOPIC_NAME
Monitoring Library 2.0, 720
RTI_Monitoring_initialize

Monitoring Library 2.0, 720
RTI_MONITORING_LOGGING_TOPIC_NAME
Monitoring Library 2.0, 721
RTI_MONITORING_PERIODIC_TOPIC_NAME
Monitoring Library 2.0, 720
rtiflat.hpp, 2314
rtps_app_id
DDS_WireProtocolQosPolicy, 1516
rtps_auto_id_kind
DDS_WireProtocolQosPolicy, 1518
rtps_host_id
DDS_WireProtocolQosPolicy, 1515
rtps_instance_id
DDS_WireProtocolQosPolicy, 1516
rtps_object_id
DDS_DataReaderProtocolQosPolicy, 844
DDS_DataWriterProtocolQosPolicy, 893
rtps_port
NDDS_Transport_UDP_WAN_CommPortsMappingInfo, 2055
rtps_protocol_version
DDS_ParticipantBuiltinTopicData, 1230
DDS_PublicationBuiltinTopicData, 1271
DDS_SubscriptionBuiltinTopicData, 1377
rtps_reliable_reader
DDS_DataReaderProtocolQosPolicy, 846
rtps_reliable_writer
DDS_DataWriterProtocolQosPolicy, 896
rtps_reserved_port_mask
DDS_WireProtocolQosPolicy, 1517
rtps_vendor_id
DDS_ParticipantBuiltinTopicData, 1230
DDS_PublicationBuiltinTopicData, 1271
DDS_SubscriptionBuiltinTopicData, 1377
rtps_well_known_ports
DDS_WireProtocolQosPolicy, 1517

Sample
connext::Sample< T >, 2184
Sample Flags, 672
DDS_DISCOVERY_SERVICE_SAMPLE, 674
DDS_INCOMPLETE_SNAPSHOT_TOPIC_QUERY, 674
DDS_INTERMEDIATE_REPLY_SEQUENCE_SAMPLE, 673
DDS_INTERMEDIATE_TOPIC_QUERY_SAMPLE, 674
DDS_LAST_SHARED_READER_QUEUE_SAMPLE, 674
DDS_REDELIVERED_SAMPLE, 673
DDS_REPLICATE_SAMPLE, 673
DDS_SampleFlag, 673
DDS_SampleFlagBits, 673
DDS_WRITER_REMOVED_BATCH_SAMPLE, 674
Sample Memory Management QoS, 321
Sample Ordering, 324
Sample States, 160
DDS_ANY_SAMPLE_STATE, 162
DDS_NOT_READ_SAMPLE_STATE, 162
DDS_READ_SAMPLE_STATE, 162
DDS_SampleStateKind, 161
DDS_SampleStateMask, 161
sample_count
DDS_DataReaderCacheStatus, 837
DDS_DataWriterCacheStatus, 890
sample_count_peak
DDS_DataReaderCacheStatus, 837
DDS_DataWriterCacheStatus, 890
sample_identity
DDS_AcknowledgmentInfo, 795
DDS_FilterSampleInfo, 1155
sample_rank
DDS_SampleInfo, 1350
sample_state
DDS_SampleInfo, 1347
sample_states
DDS_ReadConditionParams, 1289
SampleProcessor, 756
SampleRef
connext::SampleRef< T >, 2193, 2194
samples_per_app_ack
DDS_RtpsReliableReaderProtocol_t, 1319
samples_per_period
DDS_TopicQueryDispatchQosPolicy, 1405
samples_per_virtual_heartbeat
DDS_RtpsReliableWriterProtocol_t, 1325
scheduling_policy
DDS_FlowControllerProperty_t, 1157
scope
DDS_DestinationOrderQosPolicy, 933
sec
DDS_Duration_t, 1013
DDS_Time_t, 1388
secure_volatile_reader
DDS_DiscoveryConfigQosPolicy, 955
secure_volatile_writer
DDS_DiscoveryConfigQosPolicy, 954
secure_volatile_writer_publish_mode
DDS_DiscoveryConfigQosPolicy, 955
Security, 325
security_event_forwarding_level
DDS_MonitoringLoggingForwardingSettings, 1204
send_blocking
NDDS_Transport_UDPv4_Property_t, 2062
NDDS_Transport_UDPv4_WAN_Property_t, 2071
NDDS_Transport_UDPv6_Property_t, 2082
send_ping
NDDS_Transport_UDPv4_Property_t, 2063

NDDS_Transport_UDPv4_WAN_Property_t, 2072
NDDS_Transport_UDPv6_Property_t, 2083
send_reply
connext::Replier< TReq, TRep >, 2145, 2146
send_request
connext::Requester< TReq, TRep >, 2163, 2164
send_socket_buffer_size
NDDS_Transport_UDPv4_Property_t, 2058
NDDS_Transport_UDPv4_WAN_Property_t, 2068
NDDS_Transport_UDPv6_Property_t, 2078
send_window_decrease_factor
DDS_RtpsReliableWriterProtocol_t, 1336
send_window_increase_factor
DDS_RtpsReliableWriterProtocol_t, 1335
send_window_size
DDS_DataWriterProtocolStatus, 905
send_window_update_period
DDS_RtpsReliableWriterProtocol_t, 1334
Sending Large Data, 290
sent_ack_bytes
DDS_DataReaderProtocolStatus, 853
sent_ack_bytes_change
DDS_DataReaderProtocolStatus, 854
sent_ack_count
DDS_DataReaderProtocolStatus, 853
sent_ack_count_change
DDS_DataReaderProtocolStatus, 853
sent_gap_bytes
DDS_DataWriterProtocolStatus, 905
sent_gap_bytes_change
DDS_DataWriterProtocolStatus, 905
sent_gap_count
DDS_DataWriterProtocolStatus, 904
sent_gap_count_change
DDS_DataWriterProtocolStatus, 905
sent_heartbeat_bytes
DDS_DataWriterProtocolStatus, 902
sent_heartbeat_bytes_change
DDS_DataWriterProtocolStatus, 902
sent_heartbeat_count
DDS_DataWriterProtocolStatus, 901
sent_heartbeat_count_change
DDS_DataWriterProtocolStatus, 902
sent_nack_bytes
DDS_DataReaderProtocolStatus, 854
sent_nack_bytes_change
DDS_DataReaderProtocolStatus, 854
sent_nack_count
DDS_DataReaderProtocolStatus, 854
sent_nack_count_change
DDS_DataReaderProtocolStatus, 854
sent_nack_fragment_bytes
DDS_DataReaderProtocolStatus, 857
sent_nack_fragment_count
DDS_DataReaderProtocolStatus, 857
Sequence Number Support, 527
DDS_AUTO_SEQUENCE_NUMBER, 529
DDS_SEQUENCE_NUMBER_MAX, 529
DDS_SEQUENCE_NUMBER_UNKNOWN, 528
DDS_SEQUENCE_NUMBER_ZERO, 528
DDS_SequenceNumber_t, 528
Sequence Support, 757
sequence_number
WriteParams, 676
SequenceBuilders.hpp, 2315
SequenceIterator
rti::flat::SequenceIterator< E, OffsetKind >, 2201
SequenceIterator.hpp, 2324
SequenceOffsets.hpp, 2327
serialization
DDS_DynamicDataTypeProperty_t, 1139
serialize_data_to_cdr_buffer
DDSKeyedOctetsTypeSupport, 1768
DDSKeyedStringTypeSupport, 1793
DDSOctetsTypeSupport, 1814
DDSStringTypeSupport, 1858
FooTypeSupport, 1989
serialize_data_to_cdr_buffer_ex
DDSKeyedOctetsTypeSupport, 1768
DDSKeyedStringTypeSupport, 1793
DDSOctetsTypeSupport, 1814
DDSStringTypeSupport, 1859
FooTypeSupport, 1990
serialize_key_with_dispose
DDS_DataWriterProtocolQosPolicy, 895
serialized_type_object_dynamic_allocation_threshold
DDS_DomainParticipantResourceLimitsQosPolicy, 995
SERVICE, 636
DDS_ADMIN_CONSOLE_SERVICE_QOS, 637
DDS_DATABASE_INTEGRATION_SERVICE_QOS, 637
DDS_NO_SERVICE_QOS, 637
DDS_OBSERVABILITY_COLLECTOR_SERVICE_QOS, 637
DDS_PERSISTENCE_SERVICE_QOS, 637
DDS_QUEUING_SERVICE_QOS, 637
DDS_RECORDING_SERVICE_QOS, 637
DDS_REPLAY_SERVICE_QOS, 637
DDS_ROUTING_SERVICE_QOS, 637
DDS_SERVICE_QOS_POLICY_NAME, 638
DDS_ServiceQosPolicyKind, 637
DDS_WEB_INTEGRATION_SERVICE_QOS, 637
service
DDS_DataReaderQos, 865
DDS_DataWriterQos, 916
DDS_DomainParticipantQos, 976
DDS_PublicationBuiltinTopicData, 1271

DDS_SubscriptionBuiltinTopicData, 1377
service_cleanup_delay
DDS_DurabilityServiceQosPolicy, 1010
service_forwarding_level
DDS_MonitoringLoggingForwardingSettings, 1205
service_id
DDS_ServiceRequest, 1360
DDS_ServiceRequestAcceptedStatus, 1362
service_name
connext::ReplierParams< TReq, TRep >, 2154
connext::RequesterParams, 2180
connext::SimpleReplierParams< TReq, TRep >, 2211
service_request_reader
DDS_DiscoveryConfigQosPolicy, 952
service_request_writer
DDS_DiscoveryConfigQosPolicy, 950
service_request_writer_data_lifecycle
DDS_DiscoveryConfigQosPolicy, 951
service_request_writer_publish_mode
DDS_DiscoveryConfigQosPolicy, 951
ServiceRequest Built-in Topic, 496
DDS_INSTANCE_STATE_SERVICE_REQUEST_ID, 498
DDS_LOCATOR_REACHABILITY_SERVICE_REQUEST_ID, 498
DDS_MONITORING_LIBRARY_COMMAND_SERVICE_REQUEST_ID, 498
DDS_MONITORING_LIBRARY_REPLY_SERVICE_REQUEST_ID, 498
DDS_SERVICE_REQUEST_TOPIC_NAME, 499
DDS_ServiceRequest, 497
DDS_TOPIC_QUERY_SERVICE_REQUEST_ID, 498
DDS_UNKNOWN_SERVICE_REQUEST_ID, 498
set
rti::flat::PrimitiveOffset< T >, 2135
set_allocate_optional_members
DDS_TypeAllocationParams_t, 1428
set_allocate_pointers
DDS_TypeAllocationParams_t, 1427
set_attribute_mask
Activity Context, 733
set_boolean
DDS_DynamicData, 1098
set_boolean_array
DDS_DynamicData, 1112
set_boolean_seq
DDS_DynamicData, 1125
set_builtin_transport_property
NDDSTransportSupport, 2109
set_cdr_buffer
DDS_DynamicData, 1036
set_char
DDS_DynamicData, 1098
set_char_array
DDS_DynamicData, 1113
set_char_seq
DDS_DynamicData, 1125
set_complex_member
DDS_DynamicData, 1105
set_data
connext::SampleRef< T >, 2196
connext::WriteSampleRef< T >, 2225
set_default_datareader_qos
DDSDomainParticipant, 1629
DDSSubscriber, 1863
set_default_datareader_qos_with_profile
DDSDomainParticipant, 1630
DDSSubscriber, 1864
set_default_datawriter_qos
DDSDomainParticipant, 1627
DDSPublisher, 1822
set_default_datawriter_qos_with_profile
DDSDomainParticipant, 1627
DDSPublisher, 1823
set_default_flowcontroller_property
DDSDomainParticipant, 1631
set_default_library
DDSDomainParticipant, 1635
DDSDomainParticipantFactory, 1701
DDSPublisher, 1823
DDSSubscriber, 1865
set_default_params
NDDSUtilityNetworkCapture, 2119
set_default_participant_qos
DDSDomainParticipantFactory, 1698
set_default_participant_qos_with_profile
DDSDomainParticipantFactory, 1699
set_default_profile
DDSDomainParticipant, 1636
DDSDomainParticipantFactory, 1702
DDSPublisher, 1824
DDSSubscriber, 1866
set_default_publisher_qos
DDSDomainParticipant, 1640
set_default_publisher_qos_with_profile
DDSDomainParticipant, 1640
set_default_subscriber_qos
DDSDomainParticipant, 1642
set_default_subscriber_qos_with_profile
DDSDomainParticipant, 1643
set_default_topic_qos
DDSDomainParticipant, 1637
set_default_topic_qos_with_profile
DDSDomainParticipant, 1638
set_delete_optional_members
DDS_TypeDeallocationParams_t, 1498

set_delete_pointers
DDS_TypeDeallocationParams_t, 1498
set_discriminator
DDS_DynamicData, 1132
set_dns_tracker_polling_period
DDSDomainParticipant, 1681
set_double
DDS_DynamicData, 1097
set_double_array
DDS_DynamicData, 1111
set_double_seq
DDS_DynamicData, 1124
set_element
rti::flat::AbstractPrimitiveList< T >, 791
set_enabled_statuses
DDSStatusCondition, 1847
set_expression
DDSContentFilteredTopic, 1554
set_expression_parameters
DDSContentFilteredTopic, 1553
DDSMultiTopic, 1800
set_float
DDS_DynamicData, 1096
set_float_array
DDS_DynamicData, 1110
set_float_seq
DDS_DynamicData, 1123
set_handler
DDSCondition, 1545
set_info
connext::SampleRef< T >, 2196
connext::WriteSampleRef< T >, 2226
set_int8
DDS_DynamicData, 1104
set_int8_array
DDS_DynamicData, 1118
set_int8_seq
DDS_DynamicData, 1130
set_listener
DDSDataReader, 1577
DDSDataWriter, 1607
DDSDomainParticipant, 1682
DDSPublisher, 1838
DDSSubscriber, 1879
DDSTopic, 1889
set_long
DDS_DynamicData, 1093
set_long_array
DDS_DynamicData, 1107
set_long_seq
DDS_DynamicData, 1120
set_longdouble
DDS_DynamicData, 1101
set_longdouble_array
DDS_DynamicData, 1116
set_longdouble_seq
DDS_DynamicData, 1129
set_longlong
DDS_DynamicData, 1100
set_longlong_array
DDS_DynamicData, 1115
set_longlong_seq
DDS_DynamicData, 1127
set_octet
DDS_DynamicData, 1099
set_octet_array
DDS_DynamicData, 1114
set_octet_seq
DDS_DynamicData, 1126
set_output_device
NDDSConfigLogger, 2097
set_output_file
NDDSConfigLogger, 2096
set_output_file_set
NDDSConfigLogger, 2096
set_print_format
NDDSConfigLogger, 2097
set_print_format_by_log_level
NDDSConfigLogger, 2098
set_property
DDSDataReader, 1576
DDSDataWriter, 1605
DDSDomainParticipant, 1676
DDSFlowController, 1737
DDSWaitSet, 1904
set_qos
DDSDataReader, 1574
DDSDataWriter, 1604
DDSDomainParticipant, 1677
DDSDomainParticipantFactory, 1714
DDSPublisher, 1836
DDSSubscriber, 1877
DDSTopic, 1887
set_qos_with_profile
DDSDataReader, 1575
DDSDataWriter, 1605
DDSDomainParticipant, 1677
DDSPublisher, 1837
DDSSubscriber, 1878
DDSTopic, 1888
set_query_parameters
DDSQueryCondition, 1842
set_rpc_mask
NDDSConfigCompliance, 2092
set_short
DDS_DynamicData, 1094
set_short_array
DDS_DynamicData, 1108

set_short_seq
DDS_DynamicData, 1121
set_string
DDS_DynamicData, 1103
rti::flat::StringBuilder, 2215
set_thread_factory
DDSDomainParticipantFactory, 1721
set_trigger_value
DDSGuardCondition, 1741
set_uint8
DDS_DynamicData, 1105
set_uint8_array
DDS_DynamicData, 1119
set_uint8_seq
DDS_DynamicData, 1131
set_ulong
DDS_DynamicData, 1095
set_ulong_array
DDS_DynamicData, 1109
set_ulong_seq
DDS_DynamicData, 1121
set_ulonglong
DDS_DynamicData, 1100
set_ulonglong_array
DDS_DynamicData, 1115
set_ulonglong_seq
DDS_DynamicData, 1128
set_ushort
DDS_DynamicData, 1095
set_ushort_array
DDS_DynamicData, 1109
set_ushort_seq
DDS_DynamicData, 1122
set_verbosity
NDDSConfigLogger, 2095
set_verbosity_by_category
NDDSConfigLogger, 2095
set_wchar
DDS_DynamicData, 1102
set_wchar_array
DDS_DynamicData, 1117
set_wchar_seq
DDS_DynamicData, 1129
set_wstring
DDS_DynamicData, 1103
set_xtypes_mask
NDDSConfigCompliance, 2090
Shared Memory Transport, 456
NDDS_TRANSPORT_SHMEM_ADDRESS_BIT_COUNT, 460
NDDS_Transport_Shmem_create, 462
NDDS_TRANSPORT_SHMEM_GATHER_SEND_BUFFER_COUNT_MAX, 460
NDDS_TRANSPORT_SHMEM_MAJOR_AFTER_BUG_14240_FIX, 461
NDDS_TRANSPORT_SHMEM_MESSAGE_SIZE_MAX_DEFAULT, 460
NDDS_Transport_Shmem_new, 461
NDDS_TRANSPORT_SHMEM_PROPERTIES_BITMAP_DEFAULT, 460
NDDS_TRANSPORT_SHMEM_PROPERTY_DEFAULT, 461
NDDS_TRANSPORT_SHMEM_RECEIVE_BUFFER_SIZE_DEFAULT, 461
NDDS_TRANSPORT_SHMEM_RECEIVED_MESSAGE_COUNT_MAX 461
shared_secret
DDS_ParticipantTrustKeyEstablishmentAlgorithmInfo, 1238
shmem_ref_settings
DDS_DataWriterTransferModeQosPolicy, 928
shmem_ref_transfer_mode_attached_segment_allocation
DDS_DataReaderResourceLimitsQosPolicy, 883
shmem_ref_transfer_mode_max_segments
DDS_DomainParticipantResourceLimitsQosPolicy, 1004
shutdown_cleanup_period
DDS_DatabaseQosPolicy, 833
shutdown_timeout
DDS_DatabaseQosPolicy, 832
signature
DDS_ParticipantTrustAlgorithmInfo, 1235
SimpleReplier
connext::SimpleReplier< TReq, TRep >, 2207
SimpleReplierParams
connext::SimpleReplierParams< TReq, TRep >, 2210
skip_deserialization
DDS_DynamicDataTypeSerializationProperty_t, 1141
sleep
NDDSUtility, 2111
source_guid
DDS_FilterSampleInfo, 1155
DDS_SampleInfo, 1353
DDS_WriteParams_t, 1522
source_timestamp
DDS_SampleInfo, 1348
DDS_WriteParams_t, 1521
source_timestamp_resolution
DDS_BatchQosPolicy, 812
source_timestamp_tolerance
DDS_DestinationOrderQosPolicy, 933
spin
NDDSUtility, 2111
stack_size
DDS_ThreadSettings_t, 1386

start
DDSAsyncWaitSet, 1533
NDDSUtilityNetworkCapture, 2120–2122
start_with_completion_token
DDSAsyncWaitSet, 1534
status
NDDS_Transport_Interface_t, 2044
Status Kinds, 533
DDS_DATA_AVAILABLE_STATUS, 541
DDS_DATA_ON_READERS_STATUS, 541
DDS_DATA_READER_CACHE_STATUS, 546
DDS_DATA_READER_PROTOCOL_STATUS, 546
DDS_DATA_WRITER_APPLICATION_ACKNOWLEDGMENT_STATUS, 544
DDS_DATA_WRITER_CACHE_STATUS, 545
DDS_DATA_WRITER_EMPTY_VIRTUAL_WRITER_STATUS, 543
DDS_DATA_WRITER_INSTANCE_REPLACED_STATUS, 544
DDS_DATA_WRITER_PROTOCOL_STATUS, 545
DDS_DATA_WRITER_SAMPLE_REMOVED_STATUS, 546
DDS_INCONSISTENT_TOPIC_STATUS, 539
DDS_INVALID_LOCAL_ADVANCE_NOTICE_STATUS, 543
DDS_LIVELINESS_CHANGED_STATUS, 542
DDS_LIVELINESS_LOST_STATUS, 542
DDS_OFFERED_DEADLINE_MISSED_STATUS, 539
DDS_OFFERED_INCOMPATIBLE_QOS_STATUS, 540
DDS_PUBLICATION_MATCHED_STATUS, 543
DDS_RELIABLE_READER_ACTIVITY_CHANGED_STATUS, 545
DDS_RELIABLE_WRITER_CACHE_CHANGED_STATUS, 545
DDS_REQUESTED_DEADLINE_MISSED_STATUS, 540
DDS_REQUESTED_INCOMPATIBLE_QOS_STATUS, 540
DDS_SAMPLE_LOST_STATUS, 541
DDS_SAMPLE_REJECTED_STATUS, 541
DDS_SERVICE_REQUEST_ACCEPTED_STATUS, 544
DDS_STATUS_MASK_ALL, 537
DDS_STATUS_MASK_NONE, 537
DDS_StatusKind, 538
DDS_StatusMask, 537
DDS_SUBSCRIPTION_MATCHED_STATUS, 543
stop
DDSAsyncWaitSet, 1535
NDDSUtilityNetworkCapture, 2123
stop_with_completion_token
DDSAsyncWaitSet, 1535
storage_settings
DDS_DurabilityQosPolicy, 1008
stored_size
DDS_DynamicDataInfo, 1134
Stream Kinds, 166
DDS_LIVE_STREAM, 167
DDS_StreamKind, 167
DDS_StreamKindMask, 167
DDS_TOPIC_QUERY_STREAM, 167
Stream.hpp, 2338
stream_kinds
DDS_ReadConditionParams, 1289
String_Builtin_Type, 508
String Support, 757
DDS_String_alloc, 759
DDS_String_dup, 761
DDS_String_free, 762
DDS_String_replace, 761
DDS_Wstring_alloc, 762
DDS_Wstring_copy, 763
DDS_Wstring_copy_and_widen, 764
DDS_Wstring_dup, 764
DDS_Wstring_dup_and_widen, 765
DDS_Wstring_free, 765
DDS_Wstring_length, 763
string_profile
DDS_ProfileQosPolicy, 1258
subscriber
connext::ReplierParams< TReq, TRep >, 2156
connext::RequesterParams, 2182
connext::SimpleReplierParams< TReq, TRep >, 2213
Subscriber Use Cases, 401
subscriber_group_data_max_length
DDS_DomainParticipantResourceLimitsQosPolicy, 992
subscriber_key
DDS_SubscriptionBuiltinTopicData, 1376
subscriber_name
DDS_SubscriberQos, 1370
Subscribers, 130
DDS_DATAREADER_QOS_DEFAULT, 136
DDS_DATAREADER_QOS_USE_TOPIC_QOS, 137
DDS_SubscriberQos_equals, 131
print, 132
to_string, 132–136
Subscription Built-in Topics, 494
DDS_SUBSCRIPTION_TOPIC_NAME, 496
DDS_SubscriptionBuiltinTopicData, 495
Subscription Example, 393
Subscription Module, 128
subscription_handle
DDS_AcknowledgmentInfo, 795
subscription_name

DDS_DataReaderQos, 865
DDS_SubscriptionBuiltinTopicData, 1378
subscription_reader
DDS_DiscoveryConfigQosPolicy, 942
subscription_reader_resource_limits
DDS_DiscoveryConfigQosPolicy, 943
subscription_writer
DDS_DiscoveryConfigQosPolicy, 945
subscription_writer_data_lifecycle
DDS_DiscoveryConfigQosPolicy, 945
subscription_writer_publish_mode
DDS_DiscoveryConfigQosPolicy, 949
supported_mask
DDS_EndpointTrustInterceptorAlgorithmInfo, 1144
DDS_ParticipantTrustInterceptorAlgorithmInfo, 1237
DDS_TrustAlgorithmRequirements, 1426
suspend_publications
DDSPublisher, 1830
synchronization_kind
DDS_PersistentStorageSettings, 1245
System Properties, 389
SYSTEM_RESOURCE_LIMITS, 638
DDS_SYSTEMRESOURCELIMITS_QOS_POLICY_NAME, 638

tags
DDS_DataTags, 889
take
DDSKeyedOctetsDataReader, 1744
DDSKeyedStringDataReader, 1772
DDSOctetsDataReader, 1802
DDSStringDataReader, 1849
FooDataReader, 1920
take_discovery_snapshot
DDSDataReader, 1579
DDSDataWriter, 1608, 1609
DDSDomainParticipant, 1693
take_heap_snapshot
NDDSUtility, 2113
NDDSUtilityHeapMonitoring, 2115
take_instance
DDSKeyedOctetsDataReader, 1746
DDSKeyedStringDataReader, 1774
FooDataReader, 1931
take_instance_w_condition
DDSKeyedOctetsDataReader, 1746
DDSKeyedStringDataReader, 1774
FooDataReader, 1933
take_next_instance
DDSKeyedOctetsDataReader, 1747
DDSKeyedStringDataReader, 1775
FooDataReader, 1936
take_next_instance_w_condition
DDSKeyedOctetsDataReader, 1748
DDSKeyedStringDataReader, 1776
FooDataReader, 1939
take_next_sample
DDSKeyedOctetsDataReader, 1745
DDSKeyedStringDataReader, 1773
DDSOctetsDataReader, 1803
DDSStringDataReader, 1850
FooDataReader, 1928
take_replies
connext::Requester< TReq, TRep >, 2170, 2172, 2174
take_reply
connext::Requester< TReq, TRep >, 2169, 2171, 2172
take_request
connext::Replier< TReq, TRep >, 2149
take_requests
connext::Replier< TReq, TRep >, 2149
take_w_condition
DDSKeyedOctetsDataReader, 1744
DDSKeyedStringDataReader, 1772
DDSOctetsDataReader, 1802
DDSStringDataReader, 1850
FooDataReader, 1926
telemetry_data
DDS_MonitoringQosPolicy, 1212
text
NDDS_Config_LogMessage, 2040
thread
DDS_AsynchronousPublisherQosPolicy, 800
DDS_DatabaseQosPolicy, 832
DDS_EventQosPolicy, 1151
DDS_MonitoringEventDistributionSettings, 1199
DDS_MonitoringLoggingDistributionSettings, 1203
DDS_MonitoringPeriodicDistributionSettings, 1209
DDS_ReceiverPoolQosPolicy, 1294
Thread Settings, 546
DDS_THREAD_SETTINGS_CANCEL_ASYNCHRONOUS, 548
DDS_THREAD_SETTINGS_CPU_NO_ROTATION, 549
DDS_THREAD_SETTINGS_CPU_RR_ROTATION, 549
DDS_THREAD_SETTINGS_FLOATING_POINT, 548
DDS_THREAD_SETTINGS_KIND_MASK_DEFAULT, 547
DDS_THREAD_SETTINGS_PRIORITY_ENFORCE, 548
DDS_THREAD_SETTINGS_REALTIME_PRIORITY, 548
DDS_THREAD_SETTINGS_STDIO, 548
DDS_ThreadSettingsCpuRotationKind, 548
DDS_ThreadSettingsKind, 548

DDS_ThreadSettingsKindMask, 547
thread_name_prefix
DDS_AsyncWaitSetProperty_t, 804
NDDS_Transport_Property_t, 2051
thread_pool_size
DDS_AsyncWaitSetProperty_t, 803
thread_safe_write
DDS_BatchQosPolicy, 813
thread_settings
DDS_AsyncWaitSetProperty_t, 804
Throughput, 327
Time Support, 517
DDS_DURATION_AUTO, 523
DDS_DURATION_AUTO_NSEC, 523
DDS_DURATION_AUTO_SEC, 522
DDS_DURATION_INFINITE, 522
DDS_DURATION_INFINITE_NSEC, 522
DDS_DURATION_INFINITE_SEC, 522
DDS_Duration_is_auto, 521
DDS_Duration_is_infinite, 520
DDS_Duration_is_zero, 521
DDS_DURATION_ZERO, 523
DDS_DURATION_ZERO_NSEC, 523
DDS_DURATION_ZERO_SEC, 523
DDS_TIME_INVALID, 522
DDS_TIME_INVALID_NSEC, 522
DDS_TIME_INVALID_SEC, 521
DDS_Time_is_invalid, 520
DDS_Time_is_zero, 520
DDS_TIME_MAX, 521
DDS_TIME_ZERO, 518
from_micros, 519
from_millis, 519
from_nanos, 519, 520
from_seconds, 519, 520
TIME_BASED_FILTER, 639
DDS_TIMEBASEDFILTER_QOS_POLICY_NAME, 639
time_based_filter
DDS_DataReaderQos, 863
DDS_SubscriptionBuiltinTopicData, 1374
time_based_filter_dropped_sample_count
DDS_DataReaderCacheStatus, 838
timestamp
NDDS_Config_LogMessage, 2042
to_array
FooSeq, 1971
to_cdr_buffer
DDS_DynamicData, 1037
to_cdr_buffer_ex
DDS_DynamicData, 1038
to_pointer
Cookie, 671
to_string
Data Writers, 117–121
DataReaders, 147–150
DDS_DynamicData, 1038
DDS_TypeCode, 1475
DomainParticipantFactory, 45–49
DomainParticipants, 54–57
NDDSConfigVersion, 2103
Publishers, 109–113
Subscribers, 132–136
Topics, 67–71
token_bucket
DDS_FlowControllerProperty_t, 1157
tokens_added_per_period
DDS_FlowControllerTokenBucketProperty_t, 1158
tokens_leaked_per_period
DDS_FlowControllerTokenBucketProperty_t, 1159
tolerance_source_timestamp_dropped_sample_count
DDS_DataReaderCacheStatus, 838
Topic Built-in Topics, 492
DDS_TOPIC_TOPIC_NAME, 493
DDS_TopicBuiltinTopicData, 492
Topic Module, 63
Topic Queries, 155, 330
DDS_TOPIC_QUERY_SELECTION_KIND_CONTINUOUS, 160
DDS_TOPIC_QUERY_SELECTION_KIND_HISTORY_SNAPSHOT, 160
DDS_TOPIC_QUERY_SELECTION_SELECT_ALL, 160
DDS_TOPIC_QUERY_SELECTION_USE_READER_CONTENT_FILTI 160
DDS_TopicQueryData, 159
DDS_TopicQuerySelection, 158
DDS_TopicQuerySelectionKind, 159
Topic Use Cases, 396
TOPIC_DATA, 639
DDS_TOPICDATA_QOS_POLICY_NAME, 640
topic_data
DDS_PublicationBuiltinTopicData, 1269
DDS_SubscriptionBuiltinTopicData, 1375
DDS_TopicBuiltinTopicData, 1395
DDS_TopicQos, 1400
topic_data_max_length
DDS_DomainParticipantResourceLimitsQosPolicy, 991
topic_expression
DDS_TransportMulticastMapping_t, 1411
topic_name
DDS_PublicationBuiltinTopicData, 1266
DDS_SubscriptionBuiltinTopicData, 1372
DDS_TopicQueryData, 1403
topic_query_data_from_service_request
DDSTopicQueryHelper, 1896
TOPIC_QUERY_DISPATCH, 640

DDS_TOPICQUERYDISPATCH_QOS_POLICY_NAME, 641
topic_query_dispatch
DDS_DataWriterQos, 917
topic_query_guid
DDS_SampleInfo, 1354
topic_query_publication_thread
DDS_AsynchronousPublisherQosPolicy, 802
topic_query_selection
DDS_TopicQueryData, 1403
Topics, 64
DDS_DEFAULT_PRINT_FORMAT, 67
DDS_FILTERSAMPLEINFO_DEFAULT, 72
DDS_JSON_PRINT_FORMAT, 67
DDS_PRINT_FORMAT_PROPERTY_DEFAULT, 72
DDS_PrintFormatKind, 66
DDS_PrintFormatProperty, 66
DDS_TopicQos_equals, 67
DDS_XML_PRINT_FORMAT, 67
print, 67
to_string, 67–71
total_count
DDS_InconsistentTopicStatus, 1166
DDS_LivelinessLostStatus, 1179
DDS_OfferedDeadlineMissedStatus, 1220
DDS_OfferedIncompatibleQosStatus, 1221
DDS_PublicationMatchedStatus, 1274
DDS_ReliableWriterCacheEventCount, 1305
DDS_RequestedDeadlineMissedStatus, 1306
DDS_RequestedIncompatibleQosStatus, 1307
DDS_SampleLostStatus, 1355
DDS_SampleRejectedStatus, 1356
DDS_ServiceRequestAcceptedStatus, 1362
DDS_SubscriptionMatchedStatus, 1380
total_count_change
DDS_InconsistentTopicStatus, 1166
DDS_LivelinessLostStatus, 1180
DDS_OfferedDeadlineMissedStatus, 1220
DDS_OfferedIncompatibleQosStatus, 1221
DDS_PublicationMatchedStatus, 1275
DDS_ReliableWriterCacheEventCount, 1305
DDS_RequestedDeadlineMissedStatus, 1306
DDS_RequestedIncompatibleQosStatus, 1308
DDS_SampleLostStatus, 1355
DDS_SampleRejectedStatus, 1357
DDS_ServiceRequestAcceptedStatus, 1362
DDS_SubscriptionMatchedStatus, 1381
total_samples_dropped_by_instance_replacement
DDS_DataReaderCacheStatus, 840
trace_file_name
DDS_PersistentStorageSettings, 1244
traffic
NDDS_Utility_NetworkCaptureParams_t, 2088
transfer_mode
DDS_DataWriterQos, 917
Transport Address, 448
NDDS_Transport_Address_from_string, 451
NDDS_TRANSPORT_ADDRESS_INVALID, 453
NDDS_TRANSPORT_ADDRESS_INVALID_INITIALIZER, 449
NDDS_Transport_Address_is_ipv4, 452
NDDS_Transport_Address_is_multicast, 453
NDDS_Transport_Address_print, 452
NDDS_TRANSPORT_ADDRESS_STRING_BUFFER_SIZE, 450
NDDS_Transport_Address_to_string, 450
NDDS_Transport_Address_to_string_with_protocol_family_format, 451
Transport Plugins Configuration, 441
NDDS_TRANSPORT_ALLOCATION_SETTINGS_DEFAULT, 444
NDDS_TRANSPORT_ALLOCATION_SETTINGS_INCREMENTAL_CO 444
NDDS_TRANSPORT_ALLOCATION_SETTINGS_MAX_COUNT_UNL 443
NDDS_TRANSPORT_CLASSID_INVALID, 444
NDDS_TRANSPORT_CLASSID_RESERVED_RANGE, 446
NDDS_TRANSPORT_CLASSID_SHMEM, 445
NDDS_TRANSPORT_CLASSID_SHMEM_510, 445
NDDS_Transport_ClassId_t, 447
NDDS_TRANSPORT_CLASSID_TCPV4_LAN, 445
NDDS_TRANSPORT_CLASSID_TCPV4_WAN, 445
NDDS_TRANSPORT_CLASSID_TLSV4_LAN, 446
NDDS_TRANSPORT_CLASSID_TLSV4_WAN, 446
NDDS_TRANSPORT_CLASSID_UDPv4, 444
NDDS_TRANSPORT_CLASSID_UDPv4_WAN, 446
NDDS_TRANSPORT_CLASSID_UDPv6, 445
NDDS_TRANSPORT_CLASSID_UDPv6_510, 445
NDDS_TRANSPORT_CLASSNAME_TCPV4_WAN, 446
NDDS_TRANSPORT_INTERFACE_RANK_UNKNOWN, 443
NDDS_TRANSPORT_LENGTH_UNLIMITED, 443
NDDS_TRANSPORT_PORT_INVALID, 443
NDDS_Transport_Port_t, 447
NDDS_TRANSPORT_PROPERTY_BIT_BUFFER_ALWAYS_LOANED 446
NDDS_TRANSPORT_PROPERTY_GATHER_SEND_BUFFER_COUN 447
NDDS_TRANSPORT_UUID_SIZE, 443
NDDS_TRANSPORT_UUID_UNKNOWN, 443
Transport Use Cases, 409
TRANSPORT_BUILTIN, 641
DDS_TRANSPORTBUILTIN_MASK_ALL, 643
DDS_TRANSPORTBUILTIN_MASK_DEFAULT, 642
DDS_TRANSPORTBUILTIN_MASK_NONE, 642
DDS_TRANSPORTBUILTIN_QOS_POLICY_NAME,

644
DDS_TRANSPORTBUILTIN_SHMEM, 644
DDS_TRANSPORTBUILTIN_SHMEM_ALIAS, 644
DDS_TRANSPORTBUILTIN_UDPv4, 644
DDS_TRANSPORTBUILTIN_UDPv4_ALIAS, 644
DDS_TRANSPORTBUILTIN_UDPv4_WAN, 644
DDS_TRANSPORTBUILTIN_UDPv4_WAN_ALIAS, 644
DDS_TRANSPORTBUILTIN_UDPv6, 644
DDS_TRANSPORTBUILTIN_UDPv6_ALIAS, 644
DDS_TransportBuiltinKind, 643
DDS_TransportBuiltinKindMask, 643
transport_builtin
DDS_DomainParticipantQos, 974
transport_classid
NDDS_Transport_Interface_t, 2043
transport_common.h, 2228
transport_common_user.h, 2233
transport_info
DDS_ParticipantBuiltinTopicData, 1232
transport_info_list_max_length
DDS_DomainParticipantResourceLimitsQosPolicy, 1001
transport_interface_user.h, 2242
TRANSPORT_MULTICAST, 645
DDS_AUTOMATIC_TRANSPORT_MULTICAST_QOS, 646
DDS_TRANSPORTMULTICAST_QOS_POLICY_NAME, 646
DDS_TransportMulticastQosPolicyKind, 646
DDS_UNICAST_ONLY_TRANSPORT_MULTICAST_QOS, 646
TRANSPORT_MULTICAST_MAPPING, 646
DDS_TRANSPORTMULTICASTMAPPING_QOS_POLICY_NAME, 647
TRANSPORT_PRIORITY, 647
DDS_TRANSPORTPRIORITY_QOS_POLICY_NAME, 648
transport_priority
DDS_DataReaderQos, 865
DDS_DataWriterQos, 913
DDS_TopicBuiltinTopicData, 1394
DDS_TopicQos, 1402
transport_priority_mapping_high
NDDS_Transport_UDPv4_Property_t, 2063
NDDS_Transport_UDPv4_WAN_Property_t, 2072
NDDS_Transport_UDPv6_Property_t, 2083
transport_priority_mapping_low
NDDS_Transport_UDPv4_Property_t, 2063
NDDS_Transport_UDPv4_WAN_Property_t, 2072
NDDS_Transport_UDPv6_Property_t, 2083
transport_priority_mask
NDDS_Transport_UDPv4_Property_t, 2062
NDDS_Transport_UDPv4_WAN_Property_t, 2071
NDDS_Transport_UDPv6_Property_t, 2082
TRANSPORT_SELECTION, 648
DDS_TRANSPORTSELECTION_QOS_POLICY_NAME, 648
transport_selection
DDS_DataReaderQos, 864
DDS_DataWriterQos, 915
transport_shmem.h, 2245
transport_udp.h, 2249
transport_udpv4.h, 2262
transport_udpv6.h, 2273
TRANSPORT_UNICAST, 649
DDS_TRANSPORTUNICAST_QOS_POLICY_NAME, 649
transport_uuid
NDDS_Transport_Property_t, 2051
TransportAllocationSettings_t, 2218
Transports, 172, 327
transports
DDS_TransportMulticastSettings_t, 1418
DDS_TransportUnicastSettings_t, 1425
NDDS_Utility_NetworkCaptureParams_t, 2087
trigger_flow
DDSFlowController, 1738
trim_to_size
DDS_DynamicDataTypeSerializationProperty_t, 1141
trust_algorithm_info
DDS_ParticipantBuiltinTopicData, 1233
DDS_PublicationBuiltinTopicData, 1272
DDS_SubscriptionBuiltinTopicData, 1378
trust_chain
DDS_ParticipantTrustSignatureAlgorithmInfo, 1239
trust_protection_info
DDS_ParticipantBuiltinTopicData, 1233
DDS_PublicationBuiltinTopicData, 1272
DDS_SubscriptionBuiltinTopicData, 1378
type
DDS_StructMember, 1365
DDS_UnionMember, 1502
DDS_ValueMember, 1506
Type Code Support, 79
DDS_ExtensibilityKind, 89
DDS_EXTENSIBLE_EXTENSIBILITY, 90
DDS_FINAL_EXTENSIBILITY, 90
DDS_g_tc_boolean, 92
DDS_g_tc_char, 93
DDS_g_tc_double, 92
DDS_g_tc_float, 92
DDS_g_tc_int8, 93
DDS_g_tc_long, 91
DDS_g_tc_longdouble, 95
DDS_g_tc_longlong, 94
DDS_g_tc_null, 90

DDS_g_tc_octet, 93
DDS_g_tc_short, 90
DDS_g_tc_uint8, 94
DDS_g_tc_ulong, 91
DDS_g_tc_ulonglong, 94
DDS_g_tc_ushort, 91
DDS_g_tc_wchar, 95
DDS_MUTABLE_EXTENSIBILITY, 90
DDS_PRIVATE_MEMBER, 85
DDS_PUBLIC_MEMBER, 85
DDS_TCKind, 89
DDS_TK_ALIAS, 89
DDS_TK_ARRAY, 89
DDS_TK_BOOLEAN, 89
DDS_TK_CHAR, 89
DDS_TK_DOUBLE, 89
DDS_TK_ENUM, 89
DDS_TK_FLOAT, 89
DDS_TK_INT8, 89
DDS_TK_LONG, 89
DDS_TK_LONGDOUBLE, 89
DDS_TK_LONGLONG, 89
DDS_TK_NULL, 89
DDS_TK_OCTET, 89
DDS_TK_SEQUENCE, 89
DDS_TK_SHORT, 89
DDS_TK_STRING, 89
DDS_TK_STRUCT, 89
DDS_TK_UINT8, 89
DDS_TK_ULONG, 89
DDS_TK_ULONGLONG, 89
DDS_TK_UNION, 89
DDS_TK_USHORT, 89
DDS_TK_VALUE, 89
DDS_TK_WCHAR, 89
DDS_TK_WSTRING, 89
DDS_TYPE_CODE_PRINT_KIND_IDL, 88
DDS_TYPE_CODE_PRINT_KIND_JSON_SCHEMA, 88
DDS_TYPE_CODE_PRINT_KIND_XML, 88
DDS_TYPECODE_INDEX_INVALID, 83
DDS_TYPECODE_KEY_MEMBER, 86
DDS_TYPECODE_MEMBER_ID_INVALID, 83
DDS_TYPECODE_NONKEY_MEMBER, 85
DDS_TYPECODE_NONKEY_REQUIRED_MEMBER, 86
DDS_TYPECODE_NOT_BITFIELD, 83
DDS_TypeCode_PrintFormat_INITIALIZER, 87
DDS_TypeCodePrintFormatKind, 88
DDS_ValueModifier, 87
DDS_Visibility, 88
DDS_VM_ABSTRACT, 84
DDS_VM_CUSTOM, 84
DDS_VM_NONE, 84
DDS_VM_TRUNCATABLE, 84
Type System, 331
type_code
DDS_PublicationBuiltinTopicData, 1269
DDS_SubscriptionBuiltinTopicData, 1376
type_code_max_serialized_length
DDS_DomainParticipantResourceLimitsQosPolicy, 994
type_consistency
DDS_DataReaderQos, 863
DDS_SubscriptionBuiltinTopicData, 1375
TYPE_CONSISTENCY_ENFORCEMENT, 649
DDS_ALLOW_TYPE_COERCION, 651
DDS_AUTO_TYPE_COERCION, 651
DDS_DISALLOW_TYPE_COERCION, 651
DDS_TYPE_CONSISTENCY_ENFORCEMENT_QOS_POLICY_NAME 651
DDS_TypeConsistencyKind, 650
type_modifier
DDS_TypeCode, 1454
type_name
DDS_PublicationBuiltinTopicData, 1266
DDS_SubscriptionBuiltinTopicData, 1372
DDS_TopicBuiltinTopicData, 1393
type_object_max_deserialized_length
DDS_DomainParticipantResourceLimitsQosPolicy, 996
type_object_max_serialized_length
DDS_DomainParticipantResourceLimitsQosPolicy, 994
type_support
DDS_DataReaderQos, 866
DDS_DataWriterQos, 917
DDS_DomainParticipantQos, 976
TYPESUPPORT, 651
DDS_AUTO_CDR_PADDING, 652
DDS_CdrPaddingKind, 652
DDS_NOT_SET_CDR_PADDING, 652
DDS_TYPESUPPORT_QOS_POLICY_NAME, 653
DDS_ZERO_CDR_PADDING, 652

UDP Transport Plugin definitions, 454
NDDS_TRANSPORT_UDP_GATHER_SEND_BUFFER_COUNT_MAX 455
NDDS_TRANSPORT_UDP_MULTICAST_TTL_DEFAULT, 456
NDDS_Transport_UDP_Port, 456
NDDS_TRANSPORT_UDP_PROPERTIES_BITMAP_DEFAULT, 454
NDDS_TRANSPORT_UDP_RECV_SOCKET_BUFFER_SIZE_DEFAU 456
NDDS_TRANSPORT_UDP_SEND_SOCKET_BUFFER_SIZE_DEFAU 455

NDDS_TRANSPORT_UDP_SOCKET_BUFFER_SIZE_OS_DEFAULT, 455
NDDS_TRANSPORT_UDP_SOCKET_BUFFER_SIZE_OS_MAX, 455
udp_debugging_address
NDDS_Transport_Shmem_Property_t, 2054
udp_debugging_port
NDDS_Transport_Shmem_Property_t, 2055
UDPv4 Transport, 462
NDDS_TRANSPORT_UDPV4_ADDRESS_BIT_COUNT, 467
NDDS_TRANSPORT_UDPV4_BLOCKING_ALWAYS, 469
NDDS_TRANSPORT_UDPV4_BLOCKING_DEFAULT, 469
NDDS_TRANSPORT_UDPV4_BLOCKING_NEVER, 469
NDDS_Transport_UDPv4_create, 471
NDDS_Transport_UDPv4_create_from_properties_with_prefix, 472
NDDS_TRANSPORT_UDPV4_GATHER_SEND_BUFFER_COUNT_MAX_DEFAULT, 467
NDDS_TRANSPORT_UDPV4_MESSAGE_SIZE_MAX_DEFAULT, 468
NDDS_TRANSPORT_UDPV4_MULTICAST_TTL_DEFAULT, 469
NDDS_Transport_UDPv4_new, 470
NDDS_TRANSPORT_UDPV4_PAYLOAD_SIZE_MAX, 468
NDDS_TRANSPORT_UDPV4_PROPERTIES_BITMAP_DEFAULT, 467
NDDS_TRANSPORT_UDPV4_PROPERTY_DEFAULT, 469
NDDS_TRANSPORT_UDPV4_RECV_SOCKET_BUFFER_SIZE_DEFAULT, 468
NDDS_TRANSPORT_UDPV4_SEND_SOCKET_BUFFER_SIZE_DEFAULT, 468
NDDS_TRANSPORT_UDPV4_SOCKET_BUFFER_SIZE_OS_DEFAULT, 468
NDDS_TRANSPORT_UDPV4_SOCKET_BUFFER_SIZE_OS_MAX, 467
NDDS_Transport_UDPv4_string_to_address_cEA, 469
NDDS_TRANSPORT_UDPV4_WAN_ADDRESS_BIT_COUNT, 467
UDPv6 Transport, 479
NDDS_TRANSPORT_UDPV6_ADDRESS_BIT_COUNT, 483
NDDS_TRANSPORT_UDPV6_BLOCKING_ALWAYS, 485
NDDS_TRANSPORT_UDPV6_BLOCKING_NEVER, 485
NDDS_Transport_UDPv6_create, 487
NDDS_Transport_UDPv6_create_from_properties_with_prefix, 488
NDDS_TRANSPORT_UDPV6_GATHER_SEND_BUFFER_COUNT_M, 483
NDDS_TRANSPORT_UDPV6_MESSAGE_SIZE_MAX_DEFAULT, 484
NDDS_TRANSPORT_UDPV6_MULTICAST_TTL_DEFAULT, 485
NDDS_Transport_UDPv6_new, 486
NDDS_TRANSPORT_UDPV6_PAYLOAD_SIZE_MAX, 484
NDDS_TRANSPORT_UDPV6_PROPERTIES_BITMAP_DEFAULT, 483
NDDS_TRANSPORT_UDPV6_RECV_SOCKET_BUFFER_SIZE_DEFA, 484
NDDS_TRANSPORT_UDPV6_SEND_SOCKET_BUFFER_SIZE_DEFA, 484
NDDS_TRANSPORT_UDPV6_SOCKET_BUFFER_SIZE_OS_DEFAU, 484
NDDS_TRANSPORT_UDPV6_SOCKET_BUFFER_SIZE_OS_MAX, 483
NDDS_Transport_UDPV6_string_to_address_cEA, 485
unacknowledged_sample_count
DDS_ReliableWriterCacheChangedStatus, 1304
unacknowledged_sample_count_peak
DDS_ReliableWriterCacheChangedStatus, 1304
unbind_complex_member
DDS_DynamicData, 1044
unbind_type
DDS_DynamicData, 1042
uncommitted_sample_count
DDS_DataReaderProtocolStatus, 856
unicast
DDS_DataReaderQos, 864
DDS_DataWriterQos, 915
Unicast Settings, 660
unicast_enabled
NDDS_Transport_UDPv4_Property_t, 2059
NDDS_Transport_UDPv6_Property_t, 2079
unicast_locators
DDS_PublicationBuiltinTopicData, 1270
DDS_SubscriptionBuiltinTopicData, 1376
unit
DDS_TypeCode, 1469
unload_activation_library
NDDSLM, 2103
unload_profiles
DDSDomainParticipantFactory, 1716
unloan
FooSeq, 1977
unlock_condition
DDSAsyncWaitSet, 1539
unregister_contentfilter
DDSDomainParticipant, 1633

unregister_instance
DDSKeyedOctetsDataWriter, 1754, 1755
DDSKeyedStringDataWriter, 1782
FooDataWriter, 1948
unregister_instance_w_params
FooDataWriter, 1951
unregister_instance_w_timestamp
DDSKeyedOctetsDataWriter, 1755
DDSKeyedStringDataWriter, 1782, 1783
FooDataWriter, 1950
unregister_thread
DDSDomainParticipantFactory, 1716
unregister_type
DDSDynamicDataTypeSupport, 1727
DDSKeyedOctetsTypeSupport, 1766
DDSKeyedStringTypeSupport, 1791
DDSOctetsTypeSupport, 1812
DDSStringTypeSupport, 1857
FooTypeSupport, 1982
unregistered_instance_count
DDS_DataWriterCacheStatus, 891
unregistered_instance_count_peak
DDS_DataWriterCacheStatus, 892
url_profile
DDS_ProfileQosPolicy, 1258
use_42e_compatible_alignment
DDS_DynamicDataTypeSerializationProperty_t, 1140
use_checksum
NDDS_Transport_UDPv4_Property_t, 2062
NDDS_Transport_UDPv4_WAN_Property_t, 2071
User Data Type Support, 73
DDS_DATAREADER_W_DATA_CONSISTENCY_CHECK, 75
DDS_DATAWRITER_CPP, 75
DDS_HANDLE_NIL, 78
DDS_InstanceHandle_compare, 77
DDS_InstanceHandle_copy, 77
DDS_InstanceHandle_equals, 76
DDS_InstanceHandle_is_nil, 78
DDS_InstanceHandle_t, 76
DDS_TYPESUPPORT_CPP, 74
User-managed Threads, 718
DDSThreadFactory_OnSpawnedFunction, 718
USER_DATA, 653
DDS_USERDATA_QOS_POLICY_NAME, 653
user_data
DDS_DataReaderQos, 862
DDS_DataWriterQos, 914
DDS_DomainParticipantQos, 973
DDS_ParticipantBuiltinTopicData, 1230
DDS_PublicationBuiltinTopicData, 1268
DDS_SubscriptionBuiltinTopicData, 1374
user_endpoints_default_required_mask
DDS_ParticipantTrustInterceptorAlgorithmInfo, 1237
user_forwarding_level
DDS_MonitoringLoggingForwardingSettings, 1205
user_multicast_port_offset
DDS_RtpsWellKnownPorts_t, 1343
user_unicast_port_offset
DDS_RtpsWellKnownPorts_t, 1343
Using DDS:: Namespace, 433
Utilities, 195

vacuum
DDS_PersistentStorageSettings, 1245
valid_data
DDS_SampleInfo, 1351
valid_response_data
DDS_AcknowledgmentInfo, 795
value
Common types and functions, 508
DDS_AckResponseData_t, 796
DDS_Cookie_t, 830
DDS_DataRepresentationQosPolicy, 888
DDS_GroupDataQosPolicy, 1161
DDS_GUID_t, 1162
DDS_KeyedOctets, 1170
DDS_KeyedString, 1173
DDS_Octets, 1218
DDS_OwnershipStrengthQosPolicy, 1228
DDS_Property_t, 1260
DDS_PropertyQosPolicy, 1262
DDS_Tag, 1385
DDS_TopicDataQosPolicy, 1397
DDS_TransportMulticastMappingQosPolicy, 1415
DDS_TransportMulticastQosPolicy, 1416
DDS_TransportPriorityQosPolicy, 1421
DDS_TransportUnicastQosPolicy, 1424
DDS_UserDataQosPolicy, 1505
value_type
rti::flat::SequenceIterator< E, OffsetKind >, 2200
vendorId
DDS_VendorId_t, 1509
verbosity
DDS_LoggingQosPolicy, 1189
Version, 721
View States, 162
DDS_ANY_VIEW_STATE, 164
DDS_NEW_VIEW_STATE, 164
DDS_NOT_NEW_VIEW_STATE, 164
DDS_ViewStateKind, 163
DDS_ViewStateMask, 163
view_state
DDS_SampleInfo, 1348
view_states
DDS_ReadConditionParams, 1289
virtual_duplicate_dropped_sample_count

DDS_DataReaderCacheStatus, 839
virtual_guid
DDS_DataReaderProtocolQosPolicy, 844
DDS_DataWriterProtocolQosPolicy, 893
DDS_PublicationBuiltinTopicData, 1270
DDS_SubscriptionBuiltinTopicData, 1377
virtual_heartbeat_period
DDS_RtpsReliableWriterProtocol_t, 1325

wait
DDSAsyncWaitSetCompletionToken, 1542
DDSWaitSet, 1901
wait_for_acknowledgments
DDSDataWriter, 1602
DDSPublisher, 1835
wait_for_asynchronous_publishing
DDSDataWriter, 1603
DDSPublisher, 1835
wait_for_historical_data
DDSDataReader, 1564
wait_for_replies
connext::Requester< TReq, TRep >, 2167, 2168
wait_for_requests
connext::Replier< TReq, TRep >, 2148
wait_timeout
DDS_AsyncWaitSetProperty_t, 804
Waitset Use Cases, 408
waitset_property
DDS_AsyncWaitSetProperty_t, 803
warning
NDDSConfigLogger, 2099
WIRE_PROTOCOL, 654
DDS_INTEROPERABLE_RTPS_WELL_KNOWN_PORTS, 659
DDS_RTI_BACKWARDS_COMPATIBLE_RTPS_WELL_KNOWN_PORTS, 658
DDS_RTPS_AUTO_ID_FROM_IP, 658
DDS_RTPS_AUTO_ID_FROM_MAC, 658
DDS_RTPS_AUTO_ID_FROM_UUID, 658
DDS_RTPS_RESERVED_PORT_BUILTIN_MULTICAST, 658
DDS_RTPS_RESERVED_PORT_BUILTIN_UNICAST, 658
DDS_RTPS_RESERVED_PORT_MASK_ALL, 656
DDS_RTPS_RESERVED_PORT_MASK_DEFAULT, 655
DDS_RTPS_RESERVED_PORT_MASK_NONE, 656
DDS_RTPS_RESERVED_PORT_USER_MULTICAST, 658
DDS_RTPS_RESERVED_PORT_USER_UNICAST, 658
DDS_RtpsReservedPortKind, 657
DDS_RtpsReservedPortKindMask, 657
DDS_WIREPROTOCOL_QOS_POLICY_NAME, 659
DDS_WireProtocolQosPolicyAutoKind, 658
wire_protocol
DDS_DomainParticipantQos, 974
write
DDSKeyedOctetsDataWriter, 1756, 1757
DDSKeyedStringDataWriter, 1784
DDSOctetsDataWriter, 1807, 1808
DDSStringDataWriter, 1854
FooDataWriter, 1951
NDDSConfigLoggerDevice, 2101
write_w_params
DDSKeyedOctetsDataWriter, 1759, 1760
DDSKeyedStringDataWriter, 1785
DDSOctetsDataWriter, 1809, 1810
DDSStringDataWriter, 1854
FooDataWriter, 1956
write_w_timestamp
DDSKeyedOctetsDataWriter, 1758, 1759
DDSKeyedStringDataWriter, 1784, 1785
DDSOctetsDataWriter, 1808, 1809
DDSStringDataWriter, 1854
FooDataWriter, 1955
WriteParams, 674
DDS_AUTO_SAMPLE_IDENTITY, 676
DDS_SampleIdentity_equals, 675
DDS_UNKNOWN_SAMPLE_IDENTITY, 676
DDS_WRITEPARAMS_DEFAULT, 677
DDS_WriteParams_reset, 676
sequence_number, 676
writer_guid, 676
writer_attach
DDSWriterContentFilter, 1908
writer_compile
DDSWriterContentFilter, 1906
writer_compression_level
DDS_CompressionSettings_t, 827
writer_compression_threshold
DDS_CompressionSettings_t, 827
WRITER_DATA_LIFECYCLE, 654
DDS_WRITERDATALIFECYCLE_QOS_POLICY_NAME, 654
writer_data_lifecycle
DDS_DataWriterQos, 914
writer_data_tag_list_max_length
DDS_DomainParticipantResourceLimitsQosPolicy, 1002
writer_data_tag_string_max_length
DDS_DomainParticipantResourceLimitsQosPolicy, 1003
writer_depth
DDS_DurabilityQosPolicy, 1008
writer_detach
DDSWriterContentFilter, 1909

writer_evaluate
DDSWriterContentFilter, 1907
writer_finalize
DDSWriterContentFilter, 1908
writer_guid
WriteParams, 676
writer_instance_cache_allocation
DDS_PersistentStorageSettings, 1246
writer_loaned_sample_allocation
DDS_DataWriterResourceLimitsQosPolicy, 925
writer_memory_state
DDS_PersistentStorageSettings, 1248
writer_property_list_max_length
DDS_DomainParticipantResourceLimitsQosPolicy, 999
writer_property_string_max_length
DDS_DomainParticipantResourceLimitsQosPolicy, 999
writer_removed_batch_sample_dropped_sample_count
DDS_DataReaderCacheStatus, 840
writer_resource_limits
DDS_DataWriterQos, 915
writer_return_loan
DDSWriterContentFilter, 1909
writer_sample_cache_allocation
DDS_PersistentStorageSettings, 1247
writer_side_filter_optimization
DDS_ExpressionProperty, 1153
writer_user_data_max_length
DDS_DomainParticipantResourceLimitsQosPolicy, 992
WriteSample
connext::WriteSample< T >, 2220, 2221
WriteSampleRef
connext::WriteSampleRef< T >, 2224

Zero Copy Transfer Over Shared Memory, 73